

Article

Not peer-reviewed version

Stress-Testing the Reasoning Competence of LLMs with Proofs Under Minimal Formalism

[Konstantine Arkoudas](#)^{*} and Serafim Batzoglou

Posted Date: 23 April 2026

doi: 10.20944/preprints202604.1702.v1

Keywords: logical reasoning; large language models; LLMs; proofs; benchmarking



Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC, OpenAlex.

Copyright: This open access article is published under a [Creative Commons CC BY 4.0 license](#), which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

Stress-Testing the Reasoning Competence of LLMs With Proofs under Minimal Formalism [†]

Konstantine Arkoudas ^{1,*} and Serafim Batzoglou ²

¹ System 2 Labs, USA

² Seer, Redwood City, California

* Correspondence: konstantine@alum.mit.edu

[†] This article is an extended version of the 2025 EMNLP Findings paper *Stress-Testing the Reasoning Competence of LLMs With Formal Proofs*. It contains substantial new material beyond the conference version, including new tasks, results, methods, and analyses.

Abstract

We introduce PROOFGRID, a challenging benchmark suite for evaluating the reasoning competence of language models through machine-checkable proofs rather than final answers alone. PROOFGRID spans 15 tasks organized around proof writing, proof checking, proof masking, and proof gap-filling. The tasks are expressed in deliberately minimal formal notation, most notably NDL, a stripped-down language for natural deduction that fits in a short prompt and supports precise, auditable checking. As a result, correctness judgments are mechanical, reproducible, and fine-grained rather than dependent on human or LLM judges. PROOFGRID discriminates model capabilities with high resolution and covers a carefully calibrated difficulty spectrum, from foundational reasoning tests to structurally rich challenge tasks that no model currently solves, while avoiding reliance on domain knowledge, problems that can be easily outsourced to solvers, or long-context artifacts. The paper develops a detailed comparative framework for reasoning benchmarks and uses it to survey and systematize a broad and previously fragmented literature. This analysis clarifies how PROOFGRID differs from existing benchmarks not only in difficulty, but in representational choices, verification guarantees, reasoning depth, and vulnerability to shortcut-based success.

A key methodological contribution of the paper is an instrumented proof-checking pipeline that tolerates minor surface-level deviations while explicitly identifying the first substantive failure in a stretch of reasoning. This significantly sharpens measurement resolution, separating high-level proof planning from low-level execution noise, and reveals that current frontier models often possess surprisingly strong proof-strategy capabilities even when their formal proof execution is brittle. Using this pipeline, we conduct an extensive evaluation and analysis of a broad set of open and proprietary models released from late 2024 until now. Our results demonstrate strikingly rapid advances as well as sharp remaining limitations. Frontier systems now perform strongly on several foundational tasks, yet difficult PROOFGRID tasks (especially those requiring global combinatorial reasoning or low-level proof synthesis) remain far from solved. Beyond task-level accuracy, we identify widespread epistemic instability, e.g., models often generate proofs containing local logical errors that they can nevertheless recognize and reject when those same inferences are presented in isolation. We formalize this phenomenon via an Epistemic Stability Index (ESI), a quantitative measure of cross-context coherence between model outputs for different but related tasks (such as generated proofs and corresponding entailment judgments).

Finally, we complement raw accuracies with 2PL Item Response Theory (IRT) analyses, Wright maps, and a new normalized measure of task discrimination based on Fisher information and latent model abilities estimated from their responses. Taken together, these results position PROOFGRID not merely as a reasoning benchmark, but as a diagnostic framework for studying the emerging structure, limits, and internal coherence of reasoning in contemporary language models.

Keywords: logical reasoning; large language models; LLMs; LRMs; proofs; benchmarking

1. Introduction

Large language models (LLMs) have advanced rapidly, finding applications in increasingly complex problems well beyond their original scope of narrow NLP tasks like summarization, information extraction, and composition of short texts like emails. The recent emergence of the inference-compute paradigm (Snell et al. 2025) has given rise to so-called *large reasoning models* (LRMs), such as OpenAI's o-series of models along with GPT-5 and its successors; DeepSeek's R1 and related offerings; Anthropic's Claude Sonnet and Opus lines; and Google's Gemini family. These models have been trained with reinforcement learning (RL) and can perform test-time search to identify promising completions that are more likely to yield correct answers, at the expense of higher inference costs. Powered by test-time scaling and RL, language models have demonstrated dramatic improvements in logical and mathematical reasoning, with some of them surpassing human performance on highly challenging benchmarks, such as FrontierMath (Glazer et al. 2024).

However, most existing test batteries for logical reasoning are focused only on the accuracy of the answers, which are typically either binary (in response to yes/no questions); or answer keys to multiple-choice questions; or, increasingly, numbers or other $\boxed{\cdot \cdot \cdot}$ literals, such as closed-form expressions. While this approach enables automatic evaluation by simply comparing model answers against canonical solutions, it has a serious drawback: it ignores the reasoning that gives rise to the answers. This can inflate performance, because models sometimes arrive at the right answers for the wrong reasons, through plausible but ultimately spurious inferences. More fundamentally, this evaluation mode is particularly unsuitable for mathematics, where proofs are not just a means to an end but the very foundation of the discipline. A mathematical claim is accepted not because it seems likely or even because it happens to be correct in some instances, but because a rigorous and logically sound proof demonstrating its truth is given.

Therefore, emphasizing answer accuracy while overlooking the validity of the underlying reasoning is fundamentally misaligned with mathematical practice, which typically rejects a result that is derived by faulty reasoning, even if it is numerically correct. To properly assess the reasoning abilities of language models, especially in mathematics, it is essential to evaluate whether they can construct and verify proofs. PROOFGRID is a step in that direction. PROOFGRID is a benchmark suite for evaluating proof-based reasoning, requiring models not merely to produce correct answers but to construct, verify, and repair proofs expressed in a minimalist natural-deduction language (NDL) for propositional logic, and also in a pared-down formulation of pure equational logic. The benchmark comprises multiple tasks (proof writing, proof checking, proof masking, and proof gap-filling), designed to probe complementary aspects of reasoning, from generation of complex inference chains to error localization and structural reconstruction. In this paper we present and discuss all tasks in detail, evaluate 24 well-known models (both closed- and open-source) on every task, present and discuss the results, and perform extensive error analysis.

All tasks in PROOFGRID are formulated in uninterpreted formal logic. As a result, the problems are entirely abstract and thus require what psychologists call *fluid intelligence*, as opposed to *crystallized intelligence* (Sternberg, Robert 2020). Fluid intelligence refers to innate reasoning ability employed to solve novel problems, while crystallized intelligence reflects the declarative knowledge and problem-solving know-how that has accumulated through explicit instruction. Most current benchmarks allow models to leverage their vast stores of crystallized knowledge, acquired through their pretraining, which often enables them to essentially retrieve correct answers, either through brute memorization or by employing shortcuts that are sometimes valid but are other times derived from superficial patterns.

There is nothing wrong with shortcuts, of course, especially when they are based on sound theory. On the contrary, they are invaluable in practice. But when it comes to assessing native reasoning ability, they can act as confounders. If we ask a model whether a graph can be colored with 2 colors, it can answer immediately if it notices an odd-length cycle and it happens to know that a graph is 2-colorable iff it contains no odd-length cycles. By minimizing reliance on memorized content, symbolic-logic problems are much better gauges of pure reasoning competence.

To make the ramifications of this concrete with an example: o4-mini was touted as achieving 99.5% pass@1 on the 2025 AIME exam, slightly beating o3. Even more impressively, o4-mini-medium scored $\approx 22\%$ on the competition subset of FrontierMath, [outperforming the average team of human mathematicians](#) ($\approx 19\%$). On PROOFGRID, by contrast, it consistently places in the bottom half of the 24 models we have evaluated, frequently scoring less than half of the absolute score of o3,¹ and well below open-source models like DeepSeek R1 or even DeepSeek 3.1. This performance profile is more consistent with the experiences of users who report that impressive benchmark numbers often fail to translate to strong real-world performance on reasoning-intensive tasks, especially those that gain little from memorized knowledge. An advantage of abstract reasoning problems like those of PROOFGRID is that they can tease apart stronger from weaker models much more reliably. To quantify this with precision, we introduce a simple notion of spread we call DI (discriminability index), which we compute and report for every task in PROOFGRID. We also draw on IRT and statistical information theory to introduce a new normalized metric of task discrimination on select bands of latent model ability.

In addition, many existing benchmarks have become ceiling-saturated and no longer pose a serious challenge. PROOFGRID features demanding tasks that reveal the limitations of even the most capable models. We hope this will help to drive continued advances in the intersection of reasoning and language models.

Beyond the issue of ceiling effects, with leading scores essentially at 100%, many existing benchmarks cluster together even mid- and third-tier models. On MATH 500, for example, GPT-4.1 scores 87.6%, while Gemini 2.5 Pro and DeepSeek 3.1 receive 91% and 91.5%, respectively. This amounts to a spread of 3.9 points between the strongest and weakest of these three models, and a spread of only 0.5% between the top two of the three. By comparison, the corresponding scores from a randomly selected PROOFGRID task, **PL1-PW**, are 35% for GPT-4.1, 49% for DeepSeek 3.1, and 94% for Gemini 2.5 Pro, yielding a 59-point separation between first and last and a 45-point gap between the top two.

In this work we focus on reasoning tasks that demand careful and deep conceptual engagement rather than mere procedural manipulation, and cannot be straightforwardly reduced to off-the-shelf algorithms, whether implemented by external systems or coded by the model itself. This rules out puzzles like Sudoku, Towers of Hanoi, or string manipulation problems, which admit simple algorithmic solutions, and even tasks like propositional satisfiability, which can be readily outsourced to SAT solvers. By contrast, there is no standard algorithm for writing proofs in nontrivial proof systems, and while some of our equational problems could be approached with heuristics like beam search, it is not realistic for a model to engineer and execute a complete—and efficient—proof system for such problems on the fly. Even proof checking, though easily solvable by algorithm in principle, would require the one-shot design, construction, and debugging of a full software pipeline. Building such a system de novo during a reasoning task is infeasible. As a result, our benchmark compels models to perform genuine logical reasoning instead of relying on canned procedures. Moreover, by centering on proof writing, our tasks more closely reflect the kinds of reasoning challenges that arise in mathematics, science, and engineering, where the construction and justification of arguments is central. This makes PROOFGRID not only more demanding, but also more ecologically valid as a test of the reasoning abilities we ultimately want models to develop.

Some related work has probed reasoning limits using very large problems or solutions, whose size might approach or exceed model context windows (e.g., problems where the output length grows exponentially with input). With problems of such large sizes, failure is rarely surprising and not very diagnostic, as it is hard to disentangle genuine reasoning shortcomings from token-budget limits, truncation artifacts, or decoding constraints. Moreover, solutions to such problems (e.g., towers of Hanoi) tend to adhere to a fixed recursive pattern and thus stress-test symbolic bookkeeping and bandwidth more than they stress-test reasoning. By contrast, both the inputs and the expected outputs

¹ For instance, in the **PL3-PW** division of PROOFGRID, the accuracy of o4-mini is 6% compared to o3's 22%.

of PROOFGRID problems are of relatively small or modest sizes and fit easily within standard context limits, typically occupying only a few pages of text. Consequently, model failures cannot be attributed to context truncation or insufficient output budget, but instead reflect an inability to sustain long-horizon symbolic reasoning. Models must discover invariants, manage global constraints, select relevant premises under noise, reconstruct hidden combinatorial structure, and intelligently compose deep and coherent chains of inference, not simply manage context length and repetition.

In particular, our main contributions in this paper are the following:

- We introduce PROOFGRID, a rigorous reasoning benchmark that does not equate success with simple answer accuracy but instead requires models to justify and explain their deliberations by producing machine-checkable proofs. The proofs are written in extraordinarily simple formal proof languages (or, as in the case of equational logic, no formal language at all other than simple equations). The benchmark blends:
 - (a) accessible tasks serving as foundational or calibration tests, intended to gauge minimal proof-backed reasoning competence;
 - (b) more difficult tasks that are not yet quite solved by the top 2-3 models but are likely to be relatively soon;
 - (c) frontier challenges that are markedly more difficult than the rest, leaving more headroom for meaningful progress.

PROOFGRID adheres to the following design principles:

1. With the exception of the frontier challenges, all tasks, including the easier/foundational ones, have sharp discriminating power at the time of writing. This principle relates directly to the issue of benchmark saturation. There are two different notions of saturation, and they are often conflated in the community: *ceiling saturation* occurs when the strongest 2-3 models start scoring around 95% or higher. By contrast, *general or discriminative saturation* occurs when *most* models, not just the elite, start scoring above 90%. These are very different conditions. It is discriminative saturation that renders a benchmark obsolete, because it can no longer separate different levels of ability. But as long as a task effectively separates respondents of low, medium, and high ability, there is nothing objectionable about the top few models reaching 90% or higher.²
 2. Solutions do not rely on content or crystallized mathematical knowledge.
 3. Tasks are not readily solvable by code, either by external off-the-shelf solvers or by the models themselves.
 4. Tasks do not use excessively long inputs or require excessively long outputs.
 5. Tasks do not move up the difficulty ladder by introducing adversarial tricks or obscure rules, or by exploiting model quirks.
- We use the benchmark to measure and analyze the reasoning performance of a wide selection of well-known large models spanning releases from late 2024 until now. Errors in every task are grouped into categories and detailed error statistics per category are given for every model.
 - We quantify each task's discriminative power using both a simple measure we introduce based on Gini mean difference, the discriminative index (DI); as well as a new normalized score based on the Fisher information of task items and latent model abilities estimated from task results, using IRT (item response theory).
 - We introduce novel proof-manipulation tasks (proof masking and proof gap-filling) that evaluate a model's ability to reason explicitly *about* proofs, reconstruct missing inference steps, and manipulate proof structure. This extends evaluation beyond proof generation and checking to

² A school exam that gives As to most students would provide very little signal about true ability and would not be useful. But as long as an exam successfully separates students of different skill levels, it should not be dismissed just because a few students ace it. (Indeed, the top students are usually expected to earn As, otherwise the exam might be too difficult to function as a broad test of competence.)

meta-level structural understanding: do models grasp the internal structure of a reasoning artifact well enough to manipulate it successfully?

- Through extensive output analysis, we uncover and document a range of widespread behaviors, including self-contradictory explanations, shifting notions of reasoning correctness, verdict reversals mid-output, and other failure modes.
- We develop a structural framework for comparing reasoning benchmarks along orthogonal axes (input form, output structure, logical expressiveness, inferential depth, and task diversity), and use it to situate PROOFGRID within the broader reasoning-evaluation landscape.
- We develop an instrumented proof checker that tolerates minor surface-level deviations while explicitly identifying the first substantive error in a proof attempt. This allows us to disentangle failures of procedural control from failures of higher-level reasoning, enabling fairer evaluation and substantially richer error analysis than is possible with standard pass/fail checking.
- We report formal calibration results on the task of equational gap-filling, quantifying how well models' self-reported confidence aligns with empirical correctness. Using bin-level calibration gaps and expected calibration error (ECE), we find pervasive miscalibration across many models (with especially pronounced overconfidence), making naive confidence-thresholding unreliable. The resulting calibration benchmarks establish a foundation for calibration-aware training and evaluation on proof-manipulation tasks. This separates raw competence from epistemic reliability and yields a quantitative yardstick for progress on calibrated reasoning
- Beyond qualitative analysis and cataloging of reasoning pathologies, we investigate questions of reasoning *coherence across different tasks*, an analysis that is enabled by the presence of multiple related problems in the benchmark. We give a rigorous formalization of a notion that we call *epistemic stability* and provide the first quantitative results capturing the coherence of a model's reasoning across epistemic contexts, by quantifying the degree of instability between different sets of judgments (such as generated proofs and independent entailment judgments). We show empirically that even top-tier models suffer from high epistemic instability, even when their performance on a task may be high in isolation.

The rest of this paper is structured as follows. Section 2 discusses and organizes related work. Section 3 introduces the PL₁ division of PROOFGRID, simple propositional logic, which includes the following tasks (note: tasks whose names end in -c are frontier challenge tasks):

1. **PL1-PW** (proof writing, discussed in Section 3.1);
2. **PL1-PC** and **PL1-PC-c** (proof checking, Section 3.2);
3. **PL1-PM** and **PL1-PM-c** (proof masking, Section 4);
4. **PL1-GF** and **PL1-GF-c** (proof gap-filling, Section 5).

Section 6 introduces the PL₂ division, with only one task, **PL2-PW**. Section 7 describes the PL₃ division, with two tasks, **PL3-PW** and **PL3-PC**. Section 9 introduces PL₄, comprising two tasks **PL4-PW** and **PL4-PW-c**. Section 9.3 presents the equational-reasoning tasks (**EQ-PC**, **EQ-ER**, and **EQ-GF**) and corresponding results. Section 8 formalizes a notion of epistemic stability and presents related results, while Section 10 concludes. Appendix A defines NDL, a natural-deduction proof language widely used in PROOFGRID, Appendix B discusses IRT and its application to PROOFGRID, while Appendix C specifies various technical/operational details.

2. Related Work

Reasoning has emerged as a core yardstick for assessing the performance of language models. However, there are many types of reasoning and many different research directions exploring the intersection of reasoning with AI models. In this section we aim to clarify this complicated and fragmented landscape, and explain where our own work fits into the broader picture. While we try to be comprehensive, our primary focus in this paper is deductive reasoning—the bedrock of mathematics. Although copious and significant research exists on other forms of reasoning also (induction, causality, and abduction, to name a few), deduction has historically dominated the literature on AI and reasoning.

Because of the central role that deductive reasoning has played in the field, and its prominence in PROOFGRID, it will be helpful to clarify an aspect of it that is sometimes misunderstood. Deduction is often said to be a *non-ampliative* species of reasoning, meaning that there is no information in the conclusion that is not, in some sense, already contained in the premises. This supposedly contrasts with other types of reasoning, such as induction, where the conclusion conveys information beyond what is found in the premises. But this is a somewhat crude distinction that can be misleading, even though it is common both in and out of the AI community (e.g., it is embraced in the HELM framework's discussion of reasoning evaluation (Liang et al. 2023, p. 38)). It is true, of course, that every interpretation which satisfies the premises of a valid deductive argument also satisfies the conclusion, and in that strict model-theoretic sense, the conclusion cannot say anything that is not already forced by the premises. So the claim that deduction is not ampliative is true enough in the narrow technical sense of truth-conditional semantics. But it breaks down in several important respects, particularly when examined from epistemic, computational, or representational perspectives.

Epistemically and cognitively, deduction routinely yields vast amounts of information, particularly in the mathematical sciences, revealing structure, compressing information, providing explanations and warrants, and connecting distant concepts. It would be untenable to say that Euclid's theorems don't provide any information that is not already contained in his postulates, or that Fermat's last theorem doesn't tell us anything that is not already in the axioms of ZFC. Before the proof by Andrew Wiles was published in 1995, we didn't even know whether Fermat's last theorem was true; we only suspected that it was. It was the deductive reasoning of Wiles that provided justification for believing that the theorem is true, explained why it is true, connected it to elliptic curves and modular forms, and introduced powerful new techniques and rich conceptual structures.

In general, the distinction between ampliative and non-ampliative inference is much more fragile than usually assumed, as is the very distinction between deductive and nondeductive arguments. The topic is too complex to discuss in depth here, but it is worth noting that expert opinions on the subject run a wide gamut, with some argumentation theorists (who study everyday natural-language argumentation and reasoning), such as Groarke (1999), claiming that all natural-language reasoning is deductive, and that all arguments should be understood as attempts at deductive arguments. While we do not quite endorse this strong thesis here, it is important to understand that there are compelling arguments for it. In the context of AI benchmarks, the position is perhaps easier to understand when one considers problems in datasets such as MATH, like a problem asking for the roots of a quadratic equation or the area of a rectangular field. Strictly speaking, numeric answers like $\{2, 5\}$ for the equation roots or 138 for the rectangle area are not formally entailed by the problem context, because the context omits the necessary mathematical background: definitions (what is a root, what is a rectangle, etc.), relevant theorems and results (the quadratic formula, area formulas, ...), and so on. Nevertheless, the underlying reasoning remains entirely deductive. The abridged problem formulation doesn't shift the reasoning into the domain of abduction or induction, nor does it make the answer ampliative. It simply reflects the economy of expression in our representations of mathematical reasoning.

Philosophers of science such as Musgrave (2003) have defended equally strong positions regarding deduction in scientific reasoning, claiming, for example, that "the only valid arguments are deductively valid arguments, and that deductive logic is the only logic that we have or need" (Musgrave 2012) and that instances of "inductive" or "abductive" reasoning are really enthymematic deductions. Indeed, one could plausibly argue that even in what appear to be inherently inductive settings, such as applications of statistical techniques that seem to go beyond what has been strictly observed, the reasoning often remains primarily deductive. In Bayesian confirmation theory, for example, the move from prior to posterior is, mathematically speaking, an application of the probability axioms, and the update itself is a deduction made inside a richer calculus. So what might look ampliative in a bare

propositional language is non-ampliative once the relevant probabilistic assumptions are made explicit in the model specification.³

Likewise, deduction plays a key role in empirical-error-minimization methods like regression or SVM, or indeed any technique that can be viewed through the lens of statistical learning theory (SLT). This is not just because these techniques are deductively analyzed and proven to minimize error, but also because when we apply a model to a new input that we have not seen before, the reasoning is often best understood as a deduction.

The point can be illustrated with ordinary linear regression, especially when formulated in Vapnik's SLT framework. Suppose we fix a hypothesis class \mathcal{H} of linear functions $f_{\beta}(x) = \beta_0 + \beta^{\top}x$, a loss function such as squared error, and a learning rule that selects from \mathcal{H} the function minimizing empirical risk, perhaps with an added regularization term to control capacity. Suppose further that we assume the training sample and a previously unseen instance are drawn i.i.d. from a common underlying distribution, that the feature representation is stable across train and test cases, and that the target relation is appropriately approximated within the chosen hypothesis class. Then, given the observed data, it follows *deductively* which parameter vector minimizes the specified objective, which hypothesis is therefore selected by the learning rule, what prediction that hypothesis assigns to a new input, and (given the SLT machinery) what kinds of bounds relate empirical risk to expected risk as a function of sample size and hypothesis-class capacity. On this picture, the reasoning is not an irreducibly ampliative leap from observed to unobserved cases. Rather, it is a chain of deductions within a richer premise set that includes the hypothesis space, the loss function, the sampling assumptions, and the generalization theory. The genuine uncertainty lies not in the mathematical derivation of the fitted line or the associated risk bound, but in whether the assumptions underwriting that derivation are good assumptions for the problem at hand. But that is business as usual for deduction, which distinguishes between validity and soundness. Whether the premises entail the conclusion is one thing, and logic can tell us that; whether the premises are *true* (e.g., whether the test instances are indeed drawn i.i.d. from a common underlying distribution) is up to the way the world is and must be settled by empirical investigation.

In any event, whether all reasoning is ultimately deductive (or whether it can be fruitfully reconstructed as such) is a complex and contentious question, and we do not aim to take a definitive position here. Our more limited point is that the distinction between ampliative and non-ampliative reasoning (and the characterization of deduction as non-ampliative) is of questionable stability and value; and that, when scrutinized, many allegedly ampliative cases of inference are far more deductive in structure than the standard contrast suggests.

In what follows we begin with a broad overview of reasoning datasets, grouped into four categories: natural-language reasoning; mathematical and scientific reasoning; generalist reasoning; and formal reasoning. We then introduce a set of analytical dimensions that provide a finer-grained understanding of this landscape, and use these axes to classify a broad range of benchmarks, including PROOFGRID. The section concludes with a brief comparison of PROOFGRID and other major datasets.

2.1. A Bird's-Eye Survey

2.1.1. Natural-Language Reasoning

Within the larger NLP community, a major thread of work involving AI models and reasoning investigates the ability of such models to perform natural-language inference (NLI), and particularly

³ Of course, Bayesian updating by itself does not resolve the further questions of which prior to adopt, which likelihood is appropriate, or which model family to use. But that says nothing about the indispensability of induction. To make that stronger claim, one would need to show that the reasoning by which such choices are made is itself irreducibly nondeductive, rather than merely dependent on substantive assumptions or principles that are not fixed by logic alone. Uncertainty or underdetermination at the level of premises does not entail the presence of any distinctively inductive inferential mechanisms. Such uncertainty is manifested even in mathematics, e.g., consider the historical doubts regarding Euclid's fifth postulate, the axiom of choice, the continuum hypothesis and large-cardinal axioms, and even the law of the excluded middle and double negation elimination.

their ability to determine whether a set of natural-language sentences entails a target sentence. The general problem is this:

Given a set of sentences (or more generally, a text corpus) T and a hypothesis h (representing, for example, a potential answer to a question) determine whether $T \models h$, i.e., whether T entails h .

(1)

Common alternative terms for T include *context*, *theory*, *knowledge base*, or simply the set of *premises*, while the sentence h is also referred to as the—putative—*conclusion*, and sometimes as the *target sentence* or the *query*.

Note that entailment here is usually (but not always, see below) *semantic*, it is *not* formal or strictly logical entailment. In formal logic, if T entails h then it does so purely by virtue of the syntactic *form* of the sentences in T and h , not by virtue of the *meanings* of the words in those sentences. We thus say that *Tom is tall and thin* formally entails *Tom is tall* because the entailment has nothing to do with what *tall* and *thin* actually mean. We could replace those adjectives with any others and the entailment would still hold. In fact, that is how formal entailment is defined: We quantify over all possible *interpretations* of the extra-logical symbols, and we say that no matter how you interpret *Tom*, *tall*, and *thin*, i.e., for any possible way of assigning meanings (denotations) to those terms, the conclusion comes out true whenever the premises are true. By contrast, the usual textbook definition of *semantic* entailment is this: $T \models h$ iff the truth of the sentences in T “guarantees” or makes “certain” the truth of h , or equivalently, iff the falsity of h guarantees or makes certain the falsity of at least one element of T .

However, this common textbook gloss is loaded with subjective psychological notions like certainty. A crisper definition of semantic entailment can be given via possible worlds: T entails h iff in every possible world (or model) where all sentences in T are true, h is also true. This sounds very much like the definition of entailment in formal logic, but there is a crucial difference. In this definition we do not quantify over completely arbitrary interpretations of non-logical symbols. Instead, those meanings are (largely) *fixed*. Proper names like *Tom*, for example, are fixed: they are “rigid designators” that pick out the same constant element across all possible worlds. Lexical items that are not proper names (say, words like *politician* and *smoker*) are allowed some degree of cross-world variation but only in the sense that the sets of individuals answering to these descriptions (the *extensions* of these words) may vary from world to world, as one world may have a different set of politicians from another, and someone who smokes in this world might not smoke in another. But the concepts expressed by these words are uniform across all possible worlds.⁴ So the proper way to understand semantic entailment is this: once we fix the meanings of the extra-logical words *Tom*, *thin*, and *tall*, and *overweight*, in any possible world in which *Tom is thin* is true, *Tom is not overweight* is also true. Thus, *Tom is thin* semantically entails *Tom is not overweight*, even though it does not formally entail it.

Sometimes the entailment is neither formal nor semantic but rather one of implicature. Here is a textbook example: *Some students managed to pass the exam* strongly suggests that some students did not pass the exam, and so in an NLP context we might say that *Some students did not pass the exam* is entailed by *Some students managed to pass the exam*, even though it is neither semantically nor formally entailed by it. It is possible to tease out a number of other commonsensical types of entailment that are disjoint from the above kinds, but ultimately, entailment judgments in NLP are made empirically; labeled examples are annotated based on what typical annotators perceive to be consequences of a given context T . This purely descriptive framing—entailment by democracy, so to speak—is reasonable if the goal is to study entailment as an empirical phenomenon, as it functions in everyday language use. But it does not offer a normative basis for adjudicating the correctness of an inference, and this may have implications for models trained on such data. The heuristics-and-biases literature has documented a vast range of systematic reasoning fallacies to which most people (including reasoning experts) are susceptible (Tversky and Kahneman 1974). If what counts as entailment becomes conflated with what

⁴ See, for instance, Heim and Kratzer (1998, Sec. 12.3).

is commonly perceived as such, even when these perceptions are flawed, a model could score well on such benchmarks by mimicking incorrect or biased reasoning patterns.

Problem (1) originated under the name RTE (Recognizing Textual Entailment).⁵ The first RTE challenge was launched in 2005 (Dagan et al. 2009). The fourth RTE challenge, in 2008, introduced an alternative and more general formulation of the problem: Given T and h , decide whether T entails h , contradicts p , or leaves the status of p unknown—corresponding to the labels *entailment*, *contradiction*, and *neutral*. This three-way setup was later adopted and popularized by large-scale benchmarks such as SNLI (Bowman et al. 2015) and MNLI (Williams et al. 2018), helping to standardize the task in modern NLP.

Historically, NLI work studied real-world natural language, usually drawn from naturally occurring or crowdsourced text. The context T was typically a single sentence or a short paragraph at most, and the emphasis was on lexical, syntactic, and pragmatic nuances (arising, e.g., from various types of implicatures) in the relevant sentences. As already mentioned, entailment was typically defined informally,⁶ although logic-based techniques were also used from the beginning (Bos and Markert 2005; Akhmatova and Mollá 2005), albeit with underwhelming results (Bos and Markert 2006). Early logic-based efforts still dealt with short sentence pairs drawn from real-world contexts and attended closely to lexical and syntactic features (captured, e.g., by wide-coverage parsers), but they grounded entailment in logic and formal semantic representations, such as those of DRT (Discourse Representation Theory) (Kamp 1981) or MRS (Minimal Recursion Semantics) (Copestake et al. 2005). So even within mainstream NLI, there was always a thread of research that treated entailment as a rigorously defined, logic-driven relation, but applied to ecologically authentic and linguistically rich text. However, the limited success of that approach in early NLI made way for statistical models with much more shallow representations that nonetheless proved more effective in practice.

There is a relatively newer strand of NLI work that is more explicitly inspired by and tied to formal logic, exemplified by datasets like RuleTaker (Clark et al. 2021) and systems like ProofWriter (Tafjord et al. 2021). The main difference of these approaches from traditional NLI is that their data sources are expressed in *controlled* natural language (CNL), or quasi-natural language, usually templated English sentences synthetically generated from formal logical data with the help of grammars.⁷ Because the starting point is a synthetically generated formal dataset,⁸ it can be adjusted to support experiments with deeper reasoning and precisely defined entailment conditions, often grounded in subsets of logic such as stratified Datalog or various description logics. These datasets emphasize multi-step inference chains, closed-world reasoning, and deterministic label generation and evaluation, enabling fine-grained control over complexity and semantic scope. While the language is less naturalistic and rather inauthentic, the trade-off is a more transparent link between linguistic input and symbolic inference, so these datasets have been more widely used to probe the reasoning abilities of neural models. Besides RuleTaker (Clark et al. 2021) and ProofWriter (Tafjord et al. 2021), datasets in this category include ProntoQA (Saparov and He 2023), LogicInference (Ontañón et al. 2022), LogicAsker (Wan et al. 2024), LogicNLI (Tian et al. 2021), and FLD (Morishita et al. 2023). Folio (Han et al. 2024) can arguably be placed in the same group. Although its sentences are not programmatically derived from logical forms, they are nevertheless synthetic insofar as they do not occur naturally; they were manually created by human annotators based on randomly selected Wikipedia pages. The resulting

⁵ The term “NLI” started gaining wider currency after MacCartney’s influential 2009 PhD dissertation (MacCartney 2009).

⁶ Manning (2006, p.2) argued early on that a textual inference task should “adopt as the standard of inference what a human would be happy to infer from a piece of text. In particular, items would be assessed by people that are awake, careful, moderately intelligent and informed, and with reasonable document interpretation expertise, *but not by semanticists or similar academics* [our emphasis]. These people would use whatever background and real world knowledge that they usually bring to interpreting texts.”

⁷ Representative examples include *Dave is green*, *Bob is round*, *Big people are rough*, and *If someone is young and round then they are kind*. The extent to which these may be viewed as “natural” language inference problems is debatable.

⁸ This is typically a set of pairs (S, p) consisting of a collection of formal sentences S and a single sentence p —the same type of objects that we call *arguments* in PROOFGRID. (These pairs are known as *sequents* in logic, but we avoid that term here because it tends to be associated with Gentzen-style proof systems, which are not our focus in this paper.)

style is conspicuously stilted in the interest of enabling straightforward mappings to formal (FOL) notation.⁹

In general, NLI-oriented datasets (certainly those from the mainstream NLI tradition, but to a lesser extent those of the CNL variety as well), tend to focus on the inferences that are found in natural language, which are characterized by relatively shallow deductions, are strongly impacted by linguistic nuances, and incorporate commonsense—and often non-monotonic—reasoning principles such as the closed-world assumption and negation-as-failure. In this setting, AI models are viewed as “soft” reasoners (Clark et al. 2021) or soft theorem provers, simulating logical reasoning using natural language. By contrast, we will see that math- and logic-oriented datasets like PROOFGRID are usually expressed in formal symbolic notation and focus on long chains of deductions whose correctness is underwritten by strict logical and mathematical norms. They gauge the degree to which AI models can serve as “hard” reasoners or theorem provers for logic and mathematics.

While much of the work on natural-language reasoning centers around entailment, whether formally or informally defined, there are other reasoning tasks that depart from this framework. For instance, some benchmarks focus on abductive reasoning, where, given a natural-language context, the goal is to select the most plausible explanation or continuation (from a number of candidates). A representative example is HellaSwag (Zellers et al. 2019), in which models must choose the most plausible ending to a scenario from a set of options. Although this task is not about entailment, solving it requires integrating background knowledge, commonsense expectations, and causal plausibility, all of which can be said to involve (informal) inference. These benchmarks therefore probe reasoning in a broader and more intuitive sense, often drawing on patterns learned from real-world event sequences rather than strict logical inference. In contrast to deduction-focused benchmarks, the reasoning here is contextual, gradient, and plausibility-based, highlighting the diversity of what “reasoning” can mean in NLP.

2.1.2. Mathematical and Scientific Reasoning

Another class of benchmarks regularly invoked in conversations about AI models and reasoning comprises tests specifically designed to evaluate mathematical competence. Here the inputs are math problems (from various branches of mathematics and levels of difficulty) and the outputs are the respective answers. Unlike NLI, the focus is on longer chains of deductions whose correctness is underwritten by strict and explicitly formulated logical and mathematical norms, not by tacit informal conventions or pragmatics. The idea is to gauge the extent to which models can be used as “hard” logico-mathematical problem solvers. Representative samples include GSM8K (Cobbe et al. 2021), MATH (Hendrycks et al. 2021), OlympiadBench (He et al. 2024), Omni-MATH (Bofei et al. 2024), and the more demanding and recent OlymMATH (Sun et al. 2025) and FrontierMath (Glazer et al. 2024).

The problems in these benchmarks are expressed in natural language and the expected answers are what we might call *literals*, typically numbers or symbolic expressions like $x^2 - 5$. Models are expected to output their final answers in a special designated format, usually within the braces of a \LaTeX command like `\boxed{\dots}`. Questions are culled from many sources: standardized school curricula from various countries (such as K-12 textbooks); statewide and national exams (SATs, ACT, AP Calculus and Statistics, CBSE/ICSE/ISC exams in India, etc.); math competitions and Olympiads (AMC, Putnam, AIME, USAMO, IMO, and so on); materials from educational organizations like AoPS (the Art of Problem Solving); undergraduate math problem sets (e.g., MIT OCW, Harvard, etc.); crowdsourcing; and synthetic data as well.

Some of the more recent math benchmarks are remarkably challenging. FRONTIERMATH (Glazer et al. 2024), for example, introduced in late 2024 by the research institute Epoch AI with the explicit aim of evaluating AI models on advanced mathematics, comprises hundreds of original research-level

⁹ A representative FOLIO problem features the hypothesis/conclusion *Sea eel is an eel*, while the corresponding context consists of the following sentences: *All eels are fish. No fish are plants. A thing is either a plant or animal. Nothing that breathes is paper. All animals breathe. If a sea eel is either an eel or a plant, then a sea eel is an eel or an animal.*

problems across a wide range of domains, such as algebraic geometry, number theory, and category theory. These problems were created by expert mathematicians, vetted for originality, and selected for their theoretical depth and difficulty. The creators note that even seasoned mathematicians may require hours or days to solve many of them. According to Epoch AI, as of September 2025, no model reached a mean accuracy of 25% on FRONTIERMATH,¹⁰ though GPT-5 came close at 24.83%, with OpenAI's o4-mini at second place, followed by Gemini 2.5 Pro, Grok-4, and o3.¹¹ By early March 2026, the top performer was GPT-5.2, at 31.3%.

However, regardless of their difficulty level, the extent to which these datasets actually test mathematical *reasoning* is not obvious. They certainly don't do so in an explicit sense, given that they follow an outcome-based evaluation paradigm. That is, models are judged on whether they produce the correct final answer, typically a single literal value, without requiring them to provide logically correct arguments in support of those answers.¹² This is rather incongruous with the subject matter, because, since the time of Euclid, mathematics has valued proof over mere assertion. But if performance is gauged only by inspecting the final answer, models may get full credit even if they arrive at the correct result through memorization, pattern-matching, heuristic shortcuts, or statistical artifacts rather than sound reasoning. Benchmark creators do attempt to guard against guesswork (e.g., through "guessproofness" constraints (Glazer et al. 2024, p. 3) and large or intricate answers), but these measures are only partial defenses against the core issue: These benchmarks don't verify whether the model arrived at the right solution for the right reasons.¹³

A related issue with benchmarks in this category, at least as diagnostic tools for reasoning, is that they inevitably favor models with vast stores of crystallized mathematical knowledge. Large language models are trained on enormous corpora, including textbooks, research articles, lecture notes, problem banks, explanations, proofs, tutorials, and forums like StackExchange, giving them access to a breadth and depth of mathematical content far beyond what any human mathematician could hope to internalize. This gives AI models a significant advantage. Problems are built around familiar mathematical objects (polynomials, inequalities, divisibility conditions, geometric relations, and so on), where the underlying concepts, as well as common tricks and transformations, are very well-represented in the training data. Even when problems are novel or cleverly constructed or transformed, the models' ability to draw on a tremendous body of heuristics and solution templates might enable them to perform relatively well without necessarily engaging in deep reasoning, aided mostly by pattern recognition, recall, and analogy. This is especially true if the problems are heavily computational, as is the case with many of these benchmarks (including FRONTIERMATH), which allows the models to leverage their coding skills. By contrast, as mentioned in the introduction, PROOFGRID is deliberately designed to neutralize this advantage. Its problems are fully symbolic, devoid of meaningful names or context, and grounded in logic systems that offer no opportunity to exploit real-world mathematical knowledge. It thus forces models to rely primarily on fluid reasoning, by constructing and verifying proofs from abstract premises, using inferential strategies that are common in mathematical practice. This offers a clearer lens into a model's true reasoning abilities, unassisted by prior knowledge.

¹⁰ In December 2024, OpenAI claimed that o3 attained an accuracy in excess of 25%, but that claim [has been controversial](#) and Epoch AI was not able to reproduce that figure.

¹¹ These results are for the easier tiers, 1–3. The hardest tier is 4, currently led by GPT-5 with 8.33%.

¹² COT (chain-of-thought) prompting is widely used for these tests, and model outputs are (potentially long) sequences of intermediate steps culminating in a final `\boxed{...}` answer. However, this is done mostly in order to improve the question-answering performance of the models, not to evaluate the reasoning contained in the COT traces. Model performance is still judged only on whether the final answer matches the ground truth. What little preliminary work has been done in this direction so far has found that "there exists a significant disparity between answer accuracy and faithfulness of the CoT reasoning generated by LLMs, indicating that they often arrive at correct answers through incorrect reasoning" (Nguyen et al. 2024). Some newer datasets, such as NuminaMath, include a certain proportion of "proof-based problems" (Li et al. 2024, p. 12), but they do not currently support proof evaluation.

¹³ While acknowledging the technical difficulty of FRONTIERMATH, a number of Fields Medalists "pointed out that the numerical format of the questions feels somewhat contrived," with Richard Borcherds remarking, in a bit of understatement, that "these benchmark problems aren't quite the same as coming up with original proofs" (Glazer et al. 2024, p. 11).

A similar class of benchmarks target scientific problem solving. A historically important sample is the AI2 Reasoning Challenge (ARC) benchmark (Clark et al. 2018), introduced by the Allen Institute for AI and aiming to evaluate understanding and reasoning in AI models within the domain of elementary and middle school science. It contains 7,787 natural-language multiple-choice questions sourced from publicly available U.S. standardized science exams and split into two subsets:

- ARC-Easy: Questions that can often be answered by surface-level lexical cues or information retrieval techniques.
- ARC-Challenge: A curated subset of 2,590 questions that are supposed to defeat retrieval-based methods and are intended to test deeper forms of scientific and commonsense reasoning.

The questions span a broad range of science topics typically covered in grades 3–9, including the life sciences (biology, ecosystems, nutrition, human body systems, etc.); physical science (forces and motion, energy, matter, electricity); earth and space science (geology, weather, solar system); and scientific method and reasoning (experimental design, measurement, cause-effect relationships, and so on). The benchmark was explicitly designed to resist shallow statistical methods and require multi-step inference, background knowledge, and commonsense reasoning. At the time of its release in 2018, ARC represented a significant step towards evaluating genuine science-related reasoning ability in language models, but by now it has been effectively saturated, with many models achieving superhuman performance even on the Challenge subset. ARC has been superseded by the science-focused parts of relatively more recent benchmarks like BBH and Multi-MLU, although these have also been largely saturated by now.

ARC and related science-focused benchmarks bear directly on our earlier remarks regarding the distinction between deductive and nondeductive reasoning, raising important questions about how deduction should be understood and identified. Consider this ARC problem: *A student riding a bicycle observes that it moves faster on a smooth road than on a rough road. This happens because the smooth road has: (A) less gravity (B) more gravity (C) less friction [correct] (D) more friction.* This may appear to rely on simple empirical knowledge and commonsense reasoning. But if a model—or human—has internalized appropriate generalizations like “Friction impedes motion” and “Rough surfaces increase friction,” then a correct answer may be reached through a chain of deterministic (and arguably deductive) reasoning. From this perspective, the distinction between deductive and nondeductive reasoning may depend less on the task formulation and more on the implicit premises available to the test taker. This ambiguity reinforces the need for benchmarks that make the role of deduction explicit and verifiable—where the logical structure of the argument can be scrutinized and the conclusion’s validity traced to its premises.

Finally, the same issues we discussed in the context of math benchmarks apply here as well. The actual inferences of the model are not tested, only its final answer. Accordingly, these benchmarks do not directly test reasoning either, but rather question-answering ability. Likewise, crystallized knowledge plays a crucial role in successful performance in these benchmarks, and can be a confounding factor in assessing the extent to which models arrive at their answers through genuine reasoning rather than recall or statistical association. The exploitation of shallow statistical patterns has been repeatedly demonstrated for earlier and smaller transformer models, such as BERT, even on relatively abstract logical reasoning problems expressed in natural language, with, e.g., Zhang et al. (2023, 2) concluding that “instead of learning to emulate the correct reasoning function, BERT has in fact learned statistical features that inherently exist in logical reasoning problems.” But similar concerns arise for newer, larger, and more powerful models. For example, Nguyen et al. (2024) evaluated 5 families of instruction-tuned causal LLMs ranging from 7B to 175B parameters, including an early version of ChatGPT, on two multi-hop QA benchmarks. They found that the models often make the correct guess for the wrong reasons when they are asked to think step-by-step, showing a significant gap (around 16 percentage points) between answer correctness and COT correctness and concluding that “The correct final answer may not necessarily follow from faithful reasoning. We observe a significant gap between answer accuracy and reasoning faithfulness. It highlights the necessity of directly evaluating

the reasoning steps rather than solely scoring the final answers.” This is confirmed by our results, e.g., models do very well when we ask them to simply guess whether a proof is correct, but when we ask them to justify their answers with details, performance drops significantly (see Table 6). However, [Nguyen et al. \(2024\)](#) also reported that “the performance gap between the final answer and reasoning worsens as the model size increases. As the answer accuracy also increases with the model size, it suggests that the bigger models may have the knowledge of the final answer without the need to perform reasoning.” By contrast, we find that the gap shrinks for the best and largest models, because they rely less on superficial patterns and more on honest reasoning toil.

The methodological issue of assessing models indirectly, by their final answers only, extends to frontier models and to the one domain that is supposed to be the epitome of abstract deductive reasoning: mathematics. While AI models have undeniably made great strides here, the recent buzz about such models being able to solve Olympiad-level math problems is premature. A recent manual analysis of 455 carefully curated shortlist IMO (International Math Olympiad) problems from 2009 to 2023 found that “frontier models often generate incorrect solutions, that is, solutions that contain reasoning errors despite getting the final answer right.” Specifically, once the model’s full reasoning was taken into account, the study found that the conditional probability of an answer being fully correct given that the final answer was correct was 12.5% for o1, 14.3% for o3-mini, and a rather astounding 0% for o1-mini, DeepSeek R1, and Gemini 2.0 ([Mahdavi et al. 2025](#), p. 8). The authors write that “... current LLMs fall significantly short of solving challenging Olympiad-level problems and frequently fail to distinguish correct mathematical reasoning from clearly flawed solutions. Our analyses demonstrate that the occasional correct final answers provided by LLMs often result from pattern recognition or heuristic shortcuts rather than genuine mathematical reasoning. These findings underscore the substantial gap between LLM performance and human expertise in advanced mathematical reasoning and highlight the importance of developing benchmarks that prioritize the soundness of the reasoning used to arrive at an answer rather than the mere correctness of the final answers.”

2.1.3. Generalist/Hybrid Reasoning

In this category we find more recent and wider-ranging benchmarks that touch on reasoning somewhat indirectly or only in part, even though they often end up informing public claims about the reasoning abilities of AI models. Notable examples include BBH ([Suzgun et al. 2023](#)) and its successor BBEH (“Big Bench Extra Hard”) ([Google Research 2025](#)), MMLU ([Hendrycks et al. 2021](#)) and its successor MMLU Pro ([Wang et al. 2024](#)), HLE (Humanity’s Last Exam) ([Scale AI and Center for AI Safety 2025](#)), SuperGPQA ([Team et al. 2025](#)). These benchmarks aim to evaluate models across a wide array of subjects and tasks, many of which do involve reasoning to some degree. These tasks range from math and logic puzzles to legal analysis, ethics scenarios, high-school and college science, the social sciences, professional/standardized exams (SAT, GRE, LSAT, etc.), and so on. In some cases, the required inference resembles formal deduction (e.g., logic grid puzzles or syllogisms); in others, the inference relies on background knowledge, commonsense, or pragmatic reasoning.

Although these benchmarks provide a valuable broad-strokes view of model capabilities, they offer a relatively coarse-grained signal about reasoning, as they conflate different cognitive demands (e.g., domain knowledge, reading comprehension, cultural context, symbolic manipulation, and so on). Moreover, the evaluation remains outcome-based, judging models only on their final answers without assessing whether those answers are supported by correct or coherent reasoning steps. As such, these benchmarks complement more focused and diagnostic tools that aim to isolate and evaluate more precisely defined forms of reasoning.

A well-known benchmark that can be placed in this category, motivated in part by the same considerations as PROOFGRID, is the Abstraction and Reasoning Corpus for Artificial General Intelligence (ARC-AGI, or simply ARC), introduced by [Chollet \(2019\)](#). It was designed to measure human-like fluid intelligence, rather than the crystallized intelligence typically assessed by standard AI benchmarks. Because of its prominence, the rest of this section is devoted to a detailed examination of ARC.

ARC comprises few-shot visual grid-transformation tasks, where the test taker must conjecture an underlying pattern from a small number of input-output examples and successfully apply it to a novel test grid. The tasks are exclusively visual, strictly precluding not just domain-specific knowledge but also language, with the aim of relying only on fundamental *core knowledge* priors (Spelke and Kinzler 2007) like objects, numbers, actions, and space (or basic geometry).¹⁴

Humans are generally able to solve ARC tasks without much difficulty. As models scaled and researchers began applying massive test-time compute and brute-force search techniques to the original dataset, ARC-AGI-2 was introduced in early 2025 to re-establish a rigorous capability boundary (Chollet et al. 2026). ARC-AGI-2 retains the visual grid format of the original ARC-AGI (which has come to be known as ARC-AGI-1) but introduces more complex tasks (requiring, for example, a sequence of object-manipulating actions that must respect multiple rules and contextual constraints). These new tasks are not as easily solved as those in ARC-AGI-1, but humans still succeed with probability 66% when attempting to solve a randomly sampled task from the new benchmark (Chollet et al. 2026)[p. 7].

A key aim of ARC, which is shared by PROOFGRID, is to minimize the memorization and data-retrieval advantages of LLMs, and to test fluid problem-solving ability. But ARC goes further by precluding language and keeping the tasks strictly visual, with the goal of testing whether models can solve these tasks using only the few foundational priors that seem available to humans, roughly those singled out by core-knowledge theorists. Chollet speaks of “a test of human-like general intelligence” and writes that “an actionable test of general intelligence that would be fair for both humans and machines should only feature tasks that assume the four core knowledge systems listed above, and should not involve any acquired knowledge outside of these priors” (Chollet 2019)[p. 27].

ARC is an elegant benchmark that has been influential in the field, but several issues are worth noting. First, ARC claims to be a measure of general fluid intelligence, but as was acknowledged by Chollet (2019)[p. 54], no empirical evidence has been presented in support of that contention, e.g., no evidence that performance on ARC correlates meaningfully with gF , or even with g .¹⁵ This is not implausible, given that ARC has qualitative similarities to well-known nonverbal intelligence tests like RPM (Raven’s Progressive Matrices (McCallum 2003)[ch. 11]), and the latter *have* been psychometrically validated (e.g., RPM is known to load fairly strongly on g , although see Gignac (2015) for recently discovered limitations). But superficial task similarity is not enough. Psychometrically, two tasks can look alike and yet differ a lot in what drives performance: visuospatial search, working-memory load, perceptual segmentation, task-format familiarity, response-generation demands, and so on. ARC, in particular (and especially ARC-AGI-2), differs from RPM in significant ways: it is open-ended rather than multiple-choice, it requires constructing an output rather than recognizing one, it imposes heavier search demands, and uses a pixel-grid micro-world with its own idiosyncratic priors. Such

¹⁴ According to the work of developmental psychologist Spelke (2000) (see also Spelke and Kinzler (2007)), human cognition starts with a small set of evolutionarily old, domain-specific representational systems, the four main ones being objects, numbers, actions, and space (a fifth domain-specific ability, for social interaction, is acknowledged as a distinct possibility). These systems are supposed to appear early in development, be shared in some form with other animals, and provide the foundations for all subsequent learning. The core-knowledge thesis is clearly nativist, or at least anti-blank-slate, but it differs in one major respect from the more conventional nativism of Chomsky and Fodor: the role of language. The latter posit language as *the* most highly distinctive computational system of the human mind, set apart by its uniqueness, its abstract combinatorial structure, and its explanatory power. Core knowledge theorists, by contrast, tend to view language as a powerful add-on—a coordinator and amplifier of more basic cognitive systems that predate human language, insofar as they appear in infants before the development of language, and even in other animals. However, many cognitive scientists would maintain that language is so transformative that it cannot be understood as just an important add-on; on this view, language doesn’t just organize or decorate cognition, it fundamentally reorganizes it.

¹⁵ The famous g -factor (the g stands for “general intelligence”), introduced by Spearman (1904), is a latent factor inferred from positive correlations among diverse cognitive test scores (people who do well on one kind of mental task tend to do well on others, on average). It is a statistical construct from psychometrics. Spearman believed that everyone has a relatively fixed intellectual capacity, a core mental ability that plays a role in virtually every cognitive task, and that the g -factor measured precisely this ability. Gf (fluid intelligence) is a related factor, representing the capacity to reason and solve novel problems independently of previously acquired knowledge. In psychometric models (Cattell 1943, 1963; Carroll 1993), it is typically distinguished from crystallized intelligence (Gc), which reflects knowledge and skills acquired through learning and experience.

differences could substantially increase or decrease correlation with g . So while there is a plausible case for expecting some positive correlation between ARC and Gv ¹⁶, and perhaps even with g or Gf , to the best of our knowledge that remains an open empirical question.

Second, barring the use of language in a general test of intelligence (especially one that is routinely applied to language models) seems questionable, both on a priori and on empirical grounds. Language alone does not give models an unfair advantage. Language is a faculty that all neurobiologically normal humans share. It is not, by itself, crystallized intelligence, nor is it a bias; it is a key innate mental capacity that cognitive science recognizes as central to thought. It is particularly important for the most prominent forms of human reasoning, from argumentation in politics and everyday life to scientific and mathematical reasoning, which have been deeply intertwined with language¹⁷ for thousands of years, especially with reason giving. The relevant word in ancient Greek is *logos*, a rich term that can mean speech, word, account, explanation, or argument—all of them inherently linguistic objects. Likewise for *ratio*, the Latin root of *reasoning*, which also revolves around giving an account, explaining, reckoning, and justifying. These are the core practices of both everyday reasoning as well as mathematical and scientific reasoning, where one articulates premises and derives conclusions, makes and justifies claims and other inferential commitments, and explains why something is the case.

The importance of language is widely recognized in psychometric theories of intelligence and testing, even by advocates of nonverbal tests like Wechsler, who as far back as 1939 was stressing the need to include both verbal and nonverbal tasks in intelligence tests, since doing so “makes possible a comparison between a subject’s facility in using words and symbols and his ability to manipulate objects, and to perceive visual patterns” (Wechsler 1939)[p. 146]. He also made the empirical observation that “Subjects of superior intelligence generally do better on the verbal, and subjects of inferior intelligence do better on the performance (nonverbal) part of the examination” [p. 147]. A “critical finding” (Lakin and Kell 2020)[p. 531] of what is perhaps the most prominent and “widely accepted” (McGrew et al. 2023) scientific theory of intelligence, the CHC (Cattell-Horn-Carroll) theory (McGrew 2009), is that

the general reasoning factor (Gf) may be decomposed into subfactors: (1) deductive reasoning (termed sequential reasoning by Carroll and largely comprising verbal tasks), (2) quantitative reasoning (inductive or deductive reasoning with quantitative concepts), and (3) inductive reasoning (often measured with figural tasks; Carroll, 1993). A good reasoning test, then, should probably measure all three of these reasoning factors – or at least not be strongly biased toward one (Wilhelm, 2005). This fact is commonly overlooked in studies that represent fluid reasoning abilities with a single figural reasoning test, such as Raven’s Progressive Matrices test (Raven, Court, & Raven, 1977). (Lakin and Kell 2020)[p. 531]

Like RPM, ARC only taps subfactor (3) above, and even there we would claim that serious inductive reasoning (as done, e.g., in science) almost always involves language and the explicit articulation of hypotheses and evidence. The importance of language is also recognized by rival theories to CHC, such as BIS (the Berlin Intelligence Structure) (Beauducel and Kersting 2002), which holds that “measuring reasoning requires tasks that sample across the main content areas,” of which there are three: verbal, numeric, and figural; and “suggests that the diversity of content required to measure Gf consistently will require going beyond the figural tasks (e.g., Raven’s Progressive Matrices)” (Lakin and Kell

¹⁶ In the Cattell-Horn-Carroll family of psychometric theories of intelligence (Carroll 1993), the Gv factor captures the ability to perceive, analyze, manipulate, and think with visual patterns and spatial information. Walrath et al. (2020) write that “ Gv involves a range of visual processes, ranging from fairly simple visual perceptual tasks to higher-level, visual, cognitive processes. Woodcock and Mather (1989, p. 15) define Gv in part: “In Horn-Cattell theory, ‘broad visualization’ requires fluent thinking with stimuli that are visual in the mind’s eye.” They further note that “Engineers, auto mechanics, architects, nuclear physicists, sculptors, carpenters, and parts department managers all use Gv to deal with the demands of their job,” although “many writers seem to consider Gv a relatively low-level cognitive ability, more perceptual than intellectual.”

¹⁷ We understand language here in a broad sense, as a structured system of symbols governed by rules that enable the representation and communication of meaning. The only essential features are the use of discrete symbolic elements, combinatorial principles that determine how these elements can be arranged, and systematic relationships between syntax (form) and semantics.

2020)[p. 532]. Likewise, recent work on the limitations of exclusively figural tasks like RPM advises that “researchers are discouraged to include only Raven’s in an investigation, if a valid estimate of g is sought.” (Gignac 2015)[p. 71].

ARC also aims to be a test of abstraction, but it is important to be precise about what sort of abstraction mechanisms are tapped by nonverbal tests. Cognitive science has traditionally distinguished between two types of abstraction:

- low-level *perceptual abstraction*, which allows us to recognize objects across viewpoints, strip away visual noise to see that two shapes are both triangles, and so on; and
- *conceptual abstraction*, which is the process of creating higher-order categories based on invisible, functional, or relational properties, e.g., understanding that a thermometer, a clock, and a yardstick all belong to the category of measuring instruments, or understanding concepts like justice and entropy.

The sort of abstraction tested by visual tasks like ARC and RPM sits somewhere in between, and much closer to the first. It allows us to extract relational structure from perceptual representations, by identifying visual progression (e.g., increases in shape size), rotation or reflection, and patterns like same-colored regions across portions of a grid. The corresponding type of reasoning is often called *visuospatial relational reasoning* or *perceptual analogy*. While it’s true that this intermediate level of perceptual abstraction has been historically important in the testing of fluid intelligence, it is conceptual abstraction that is central to science and mathematics and is considered to be the defining cognitive capacity behind theoretical reasoning. As already mentioned, conceptual abstraction involves the formation of general ideas and principles that are detached from any particular sensory inputs. They range from ordinary concepts like cause and effect or kinship (e.g., the notion of an uncle) to scientific quantities like acceleration, the idea of natural selection in biology, a random variable in statistics, a semigroup in mathematics, and so on. It is these linguistically scaffolded abstractions that allow generalization across widely different contexts and enable sophisticated reasoning, explanation, and prediction. None of these are exercised by visual tests. So, while success on visual tasks might well provide some signal on intelligence, it should not be understood as indicative of general or high-level abstraction ability. The latter goes much farther than what is visually given, by relying on language to represent categories that depend on inferential roles rather than perceptual relations.¹⁸

It might be objected that ARC was never intended to test high-level reasoning or abstraction; what has made it notable is that powerful AI models have found it challenging even though it is a fairly simple test that many ordinary people are able to pass. However, ARC aims to be a test of “abstraction and reasoning” simpliciter, not a test of perceptual abstraction and visuospatial reasoning. The paper that introduced the benchmark speaks of a general measure of intelligence, and the relevant Kaggle competition presents itself as a test to find the next frontier of AGI, implying human-level mastery across all cognitive domains. As for the observation that models are struggling with something that even children can do, and the resulting temptation to conclude that models are incapable of reasoning, it should be kept in mind that a 5-year-old human has a 500-million-year evolutionary head start in visual processing. The human brain dedicates a massive portion of its cortex to automatic, unconscious visual grouping (Gestalt principles like proximity, similarity, and closure). For humans, “seeing” the grid and grouping the pixels requires zero effort. This raises the question of whether the exclusion of

¹⁸ Modern developmental science indicates that conceptual abstraction is produced by different learning systems than perceptual abstraction, emerges at different developmental stages, and is subserved by different mechanisms, requiring cross-modal integration, lexicalization, executive function, and the learning of rule-based categories. According to Sloutsky (2010), the brain contains two distinct category-learning systems. One is a similarity-based system characterized by compression, which learns from perceptual correlations, operates on many overlapping features, is present early in development, and is tied to the inferotemporal cortex and perceptual systems; this is the system that learns “dense” categories like dogs and cats. The other is a rule-based system characterized by selection, which selects specific diagnostic dimensions, supports rules, requires attention and executive control, is tied to the prefrontal cortex, and develops later; this is the system that learns “sparse” categories, like prime numbers and moral agency. Sloutsky (2010) points out that sparse categories tend to depend on unobservable properties and require selective attention and rule learning; and that abstract reasoning in mathematics and science depends on representational resources that go far beyond visual structure: the ability to formulate and manipulate concepts whose defining properties are unobservable, inferred, and linguistically articulated.

language in order to be fair to humans might not tip the scales the other way and make the test unfair to language models.

A brief historical aside might be helpful here: nonverbal intelligence tests originated in the 1800s because scientists were curious about the cognitive abilities of language-deprived people who could not speak. During World War I, the US Army needed “methods to assess abilities of foreign born and illiterate military recruits” and a committee to design such methods was formed (McCallum 2003)[p. 5]. Later in the 20th century, fairness came to be an increasingly prominent motivation behind nonverbal intelligence tests, some of which (such as the Universal Nonverbal Intelligence Test (Bracken and McCallum 1998)) had fairness as their “overarching goal” (McCallum 2003)[p. 88]. After all, if one hardly speaks English, it is not fair to give them an intelligence test that requires fluency in English. By the same token, however, it is not fair to give a visual intelligence test to a subject who is visually impaired. And to this day, the perceptual abilities of language models, even multi-modal ones, remain relatively poor, lagging far behind their language skills. This is largely because transformer-based models, which were designed for sequence statistics, lack the geometric and physical priors encoded in the human brain, i.e., they lack most of the core-knowledge priors that humans have (objects, actions, and space).¹⁹ Given that ARC seems heavily gated by perceptual access to the relevant tasks, visual ability may be acting as a confound. This is essentially the argument made by recent work (Wang et al. 2026), which found that models frequently fail on ARC tasks before reasoning even begins, simply because they are unable to reliably identify objects, shapes, and spatial relations (something which is effortless for humans), and therefore end up misinterpreting the visual input. Manual error analysis carried out by the authors suggests that 80% of model failures are due to perceptual errors. They also found that when the task is split into a 2-stage pipeline, a perception stage that takes the image and outputs a natural-language description of it, followed by a “reasoning” stage that takes the natural-language description and reasons with it, performance improves significantly when the results of the first (perceptual) step are improved, e.g., simply by using a stronger visual model, while keeping the reasoning model fixed. The authors thus argue that ARC results can overestimate the reasoning gap between humans and AI, because humans have much stronger visual object recognition and perceptual grouping.

These considerations suggest a divergence between the methodological guidelines in the paper that introduced ARC and the empirical benchmark ecosystem that grew around it. Chollet (2019)[p. 45] writes that a general benchmark of intelligence “should work for both humans and machines, fairly, by only assuming the same priors as possessed by humans (e.g. Core Knowledge) and only requiring a human-sized amount of practice time or training data” and “we recommend only comparing the intelligence of systems that assume a sufficiently similar set of priors” (p. 44). But in practice the test is routinely used to compare the intelligence of humans and transformer-based models, which have radically different priors.

To be clear, we are not claiming that all reasoning is linguistic, nor that ARC fails to test reasoning altogether. There are well-established non-linguistic forms of reasoning, visuospatial reasoning being one of them, and ARC does appear to tap abilities in this domain; we would not be surprised if ARC performance is shown to correlate with g_V . What remains empirically unclear is exactly which abilities ARC measures and how strongly performance on the benchmark correlates with broader measures of intelligence, and particularly which of the prominent CHC factors it loads on most strongly. Our claim is that a vast and very important portion of human reasoning and abstraction is conceptual or theoretical: it concerns categories, relations, and entities that are not directly perceptible and whose representation and manipulation are inextricably linked with language. Because nonverbal tests like

¹⁹ Just as they lack grounding in the physical world, with no direct experience of physical interactions such as movement and object manipulation, and no experience of temporal continuity. A baby learns about object persistence by interacting with the physical world, not by ingesting a static Internet snapshot.

ARC exclude language by design and operate over fully observable structures, they leave this large and very significant class of reasoning capabilities untested.²⁰

Finally, even though ARC was designed as a test that does not use language, largely in an effort to eliminate reliance on memorized patterns and to measure fluid intelligence, many of the systems that perform best on ARC are neuro-symbolic pipelines that do not perform simple end-to-end prediction of the output grid (Chollet et al. 2025). Instead, they parse the grid into objects, regions, and colors; they express candidate solutions in a DSL (domain-specific language) or restricted Python-like program space; they explicitly search over compositions of primitives, keeping only candidates that fit the training examples; and then run the resulting program on the test input. This process effectively maps these problems back into a symbolic *language* (Python or DSLs) and relies on formal syntax, semantics, and logic to synthesize a solution. And given that models are pretrained on terabytes of Python code, this allows them to lean on the massive linguistic and structural priors they memorized during pretraining, which seems at odds with a core-knowledge test. While it's true that the tasks themselves are purely visual, if the solver is a giant model pretrained on vast bodies of Python, iteration patterns, grid manipulation idioms, and so on, and relies on that know-how during problem solving, then language priors re-enter through the back door.

2.1.4. Formal Reasoning

The final category comprises benchmarks whose input problems are fully formal, expressed in symbolic logic or formal mathematical notation. These benchmarks are designed to probe reasoning in a highly controlled setting, abstracted away from the vagaries of natural language or domain-specific conventions.

Unlike natural-language or generalist benchmarks, which often involve a mix of cognitive competencies and inferential strategies, formal reasoning benchmarks aim to isolate and evaluate deductive capacity in its purest form. The tasks typically involve proving or refuting precise claims, or applying logical rules within a constrained formal system. Most benchmarks in the category comprise contentful but formally structured math problems (as in MiniF2F (Zheng et al. 2022)); datasets like PROOFGRID that aim to capture content-agnostic reasoning are relatively rare (Rosetta (Baek, Shaun Lee et al. 2025) is arguably another example, but see our comments below).

A key advantage of such benchmarks is that they are interpretable and verifiable: because the logic is fully specified, it is possible to determine unambiguously whether a response is correct and to assess the structure of the reasoning that led to it. This makes them especially well-suited for fine-grained diagnostic evaluation of a model's inferential abilities. Moreover, the absence of linguistic ambiguity or commonsense knowledge demands allows these benchmarks to more cleanly separate reasoning skill from linguistic fluency or background knowledge.

That said, formal reasoning benchmarks introduce their own complications. They tend to require a high degree of familiarity with the underlying logic or proof calculus, which makes them less accessible to general-purpose models or to researchers unfamiliar with the target formalism. Some benchmarks are grounded in complex systems like higher-order logic or dependent type theory, or rely on interactive proof protocols (such as Lean-style tactic traces) that complicate the relationship between proof generation and proof explanation, as we discuss later. PROOFGRID mitigates these challenges by adopting a minimalist *natural deduction* language, featuring the same general style of

²⁰ An analogy between reasoning and humor might help to clarify our position. Language is not strictly necessary for reasoning, in the same way that it is not necessary for humor—consider Chaplin's silent films, slapstick, timing, visual surprise, incongruous or absurd gesture (Monty Python's ministry of silly walks), and so on. At the same time, the most effective and intelligent humor is conveyed with the use of language, which enables irony, parody, understatement, narrative framing, double meaning, register shifts, self-reference, and the management of multiple perspectives at once. We would likewise claim that the most sophisticated forms of reasoning are enabled by and expressed through language and symbolism, which allow for explicit premises and conclusions, scope distinctions, quantification, modality, counterfactuals, recursive embeddings, and above all the articulation of why something follows. Taking silent pratfalls as the essence of comedy would be like taking grid transformations as the essence of reasoning.

reasoning commonly found in mathematics and logic textbooks. That language is fully described in a concise self-contained prompt.

All in all, formal reasoning benchmarks play an essential role in the ecosystem of reasoning evaluation, offering a uniquely rigorous lens on deductive competence.

2.2. Reasoning Dimensions

To better understand the diversity of reasoning datasets and benchmarks in NLP and AI in general, we position these along three main orthogonal axes:

1. an *input representation* axis;
2. an *output representation* axis; and a
3. a *content* or *crystallized knowledge* axis.

These axes reflect conceptually independent dimensions that shape the kinds of reasoning that a benchmark can evaluate.²¹

The first axis captures the form in which the inputs of a dataset are expressed. This could be naturalistic human language, or synthetically generated natural language, or formal symbolic notation, or something in between. This choice usually reflects the motivation behind the benchmark. Datasets like SNLI and MNLi are squarely in the mainstream NLI tradition and are thus invariably focused on natural language. Their main aim is to test whether language models can simulate human-like reasoning from textual input. Towards the middle of this axis we find datasets like RuleTaker and FLD, which are formulated in the sort of controlled natural language (CNL) described earlier: templated English sentences generated from symbolic rules. This results in data that is expressed in natural language but is formulaic, with very limited linguistic diversity, and semantically aligned with formal logic. Finally, at the right end of this axis we find datasets like Rosetta, whose problems are expressed entirely in formal symbolic notation, using logics of varying representational power (like propositional, first-order, or higher-order logic).

The second axis characterizes the structure and complexity of the required model outputs. On one end we have tasks that demand only a binary decision, such as whether a hypothesis follows from a context (e.g., entailment vs. non-entailment), or the selection of a single correct answer from multiple choices. These setups are straightforward to evaluate and model, but offer limited insight into the actual reasoning that models perform. On the other end of the spectrum are tasks requiring more structured reasoning artifacts, such as chains of deductive steps, natural-language justifications (e.g., COT traces), entailment trees (as in EntailmentBank), or even full formal proofs (as in PROOFGRID). These richer outputs enable finer-grained analysis of a model's reasoning process and can support interpretability, proof checking, and compositional generalization, but typically demand more supervision and are harder to generate.

The third axis characterizes reasoning benchmarks in terms of their content demands, capturing the type and extent of domain knowledge required to solve them. At one end of this spectrum we have content-free benchmarks, such as PROOFGRID, which involve purely symbolic structural reasoning tasks in uninterpreted settings. These tasks rely on the internal structure of the inference tasks and require little external knowledge. Farther along this axis we find benchmarks grounded in commonsense or everyday content, such as SNLI or bAbI (Weston et al. 2016), where understanding is based on familiar types of objects, events, and social conventions. Moving along, we have benchmarks

²¹ The axes are not continuous, but orthogonality is more important than continuity in a conceptual framework of this kind. A more granular analysis could refine each independent dimension into a categorical or ordinal feature with a specific set of interpretable values. We have already taken preliminary steps in this direction, e.g., we introduce a number of discrete levels for the output-representation axis: *binary*, *multiple-choice*, *literal*, *chain-of-thought*, *entailment tree*, *Python code*, and *formal proof*. We leave the development of a finer-grained and exhaustive classification scheme to future work. Note also that while we treat the input representation and content axes as conceptually orthogonal, in practice they are not always independent. In particular, benchmarks with natural language inputs almost inevitably carry *some* semantic content, even when synthetically constructed. Although it is possible to design grammatically well-formed yet semantically vacuous sentences in Jabberwocky style (e.g., "Every glorb is a zibble"), such examples are highly contrived. Thus, while the axes are formally distinct, they may be somewhat statistically correlated in practice.

involving scientific content, like GPQA (Rein et al. 2023) or ARC (Challenge Set) (Clark et al. 2018), which require reasoning over concepts and relations introduced by scientific theories, and are thus accessible only to those with some scientific education. At the far end of the axis we have pure math benchmarks such as MATH, GSM8K, or Lean-based proof datasets, where the content is fully mathematical, and reasoning requires both mathematical knowledge and symbolic manipulation. Although diverse, these benchmarks fall along a coherent continuum that reflects increasing degrees of semantic specificity, abstraction, and crystallized domain knowledge. Framing this as a content axis allows us to compare reasoning tasks not just by how they are formulated, but also by what kind of knowledge they demand.

In addition to these three primary axes, we identify three further dimensions that shed light on the internal structure and reasoning demands of a benchmark: the expressiveness of the underlying formalism, the range of inferential mechanisms required, and the diversity of its tasks. These axes are secondary and only relevant for datasets that are grounded in formal logic or in natural language rendered from logical forms.

Formalism Expressiveness: For benchmarks that are defined in, or translated from, a formal logic, including both fully symbolic datasets like PROOFGRID and CNL benchmarks like RuleTaker, a key consideration is the expressive power of the underlying logical system. This axis captures the range of propositions that the formalism can represent and thereby constrains the structural and semantic complexity of the problem instances. For example, RuleTaker is grounded in stratified Datalog, which supports recursive Horn clauses and limited negation but disallows full disjunction, unrestricted negation, quantifier alternation, and function symbols. Other benchmarks might be based on propositional logic, first-order logic fragments, equational logic, description logics, epistemic, modal, temporal, or spatial logics, or more powerful systems like higher-order logic or dependent type theory. Each of these formalisms permits a different set of logical constructs (such as quantifiers, equality, modal operators, or arithmetic) which in turn determine the space of conceivable problems. Two benchmarks may both use formal representations yet differ dramatically in the depth and nuance of statements they can encode, and thus on the types of reasoning that they can test, depending on where they lie on this dimension.

Inferential Repertoire: This axis captures the range and sophistication of inference mechanisms that a model must deploy to explain and justify its answers to a benchmark's problems. It is mostly relevant for benchmarks that require models to produce *structured reasoning artifacts* that are amenable to automated analysis, such as entailment trees or formal proofs. For benchmarks that only require final answers, the model's reasoning process remains opaque and this axis becomes inert; without observable reasoning traces, it is not possible to determine what inference methods were used, short of speculating about latent cognitive or model-internal dynamics. The same applies to benchmarks that elicit *unstructured* reasoning artifacts, such as chain-of-thought traces expressed in arbitrary natural language. Even if such traces are generated during inference, they are typically not evaluated and are therefore of little help in analyzing the underlying reasoning mechanisms.

When structured reasoning outputs *are* generated, this axis ranges from benchmarks that require only simple chaining or forward inference using a small set of rudimentary rules (such as modus ponens and conjunction elimination) to tasks that demand complex reasoning strategies such as hypothetical proof, arbitrarily nested case analyses, universal generalization, reasoning by contradiction, existential instantiation, and various forms of mathematical induction (natural, structural, strong, etc.).

While this dimension often correlates with the expressiveness of the formalism, it is not reducible to it. A benchmark may employ an expressive logic in theory while exercising only a weak fragment in practice. In such cases, the inferential repertoire axis provides an independent informative lens on the range of reasoning tested by a benchmark.

Task Diversity: Another important dimension for characterizing a reasoning benchmark is task breadth—the number and variety of distinct reasoning tasks it includes. Some benchmarks are narrowly focused, posing a single core problem, such as entailment: “Does hypothesis h follow from context T ?” Even within that core task, variations can arise depending on the output format (e.g., binary label vs. a supporting explanation in COT style), but the central reasoning objective remains the same. Other benchmarks expand the task scope by supporting or requiring multiple types of reasoning, each probing different varieties of inferential ability. For example, in addition to entailment verification, a benchmark might include tasks like countermodel generation (demonstrating that $T \not\models h$), error diagnosis (identifying why a hypothesis fails), or hypothesis repair (suggesting minimal modifications that ensure $T \models h$). The number and diversity of these subtasks can be treated as a measure of a benchmark’s functional richness.

Greater task diversity broadens what a dataset can reveal and enables finer-grained assessment of a model’s reasoning, but it also complicates evaluation and interpretation. Benchmarks with many distinct subtasks can suffer from a lack of cohesion, potentially conflating heterogeneous cognitive demands. This can make it difficult to attribute model success or failure to specific competencies. Finally, aggregating performance across many diverse tasks tends to lack a principled foundation, raising questions about how to interpret a single combined score. As such, while broad task coverage offers diagnostic benefits, there is also value in relatively focused benchmarks with tightly defined reasoning objectives. Diversity is most effective when the subtasks are conceptually coherent and mutually reinforcing, rather than arbitrarily bundled together. This axis thus highlights a tension between diagnostic richness and interpretive clarity.

Together, these auxiliary axes complement the existing framework and offer finer-grained tools for analyzing what makes a benchmark challenging.

Table 1 lists a number of datasets that are intended, either entirely or in part, to test the reasoning abilities of AI models. The datasets are characterized according to the dimensions discussed above, along with a few additional attributes such as the maximum reasoning depth required (a rough ceiling on problem difficulty); current saturation status; and whether a model’s reasoning is verified programmatically. A NA entry (—) under **Size** means that the dataset has no fixed size. This usually means that the authors provide a script that can generate any number of data points, often set to a certain default value (e.g., 5K in the case of LogicInference (Ontañón et al. 2022, p. 10)). In the remainder of this section we discuss the various entries in Table 1.

ProntoQA’s logic is positive Horn logic, which is reducible to Datalog over unary predicates. The few negated properties in ProntoQA are just specially named predicates; they do not introduce or need the semantic machinery—and potential non-monotonicity—of true stratified negation. Accordingly, the only reasoning mechanism that is needed for ProntoQA proofs is modus ponens.

Although the ENTAILMENTBANK paper (Dalvi et al. 2021) outlines three tasks, these are all varying scales of the same problem, entailment tree construction, so we have consolidated them under a single task designation. We have listed PROOFGRID’s task number as 4/15 to indicate that there are 4 general types of tasks but with 15 separate datasets/instantiations. We report MMLU as comprising 57 tasks, following the paper that introduced the benchmark (Hendrycks et al. 2021), but note that these are not 57 distinct task types in terms of input/output format or basic problem-solving requirements. Instead, they all share the same basic task structure, namely, multiple-choice question answering (MCQA), typically with 4 answer choices. The variation across the 57 datasets lies entirely in the domain or subject matter (STEM questions, social-science questions, anatomy, logical fallacies, moral disputes, and so on).²² A more recent version of this benchmark, MMLU-Pro (Wang et al. 2024), was introduced in 2024 with the aim of filtering out trivial questions from MMLU; reducing the number of categories from 57 to 14; incorporating new questions from TheoremQA (Chen et al. 2023), SciBench, and the STEM website; and adding several additional distractor options to multiple-choice questions.

²² It’s an unfortunate trend that *task* is becoming increasingly conflated with *subject matter*. As another example, the paper that introduced the HLE benchmark (Humanity’s Last Exam) characterizes benchmarks as “collections of questions which assess model performance on tasks such as math, programming, or biology” (Scale AI and Center for AI Safety 2025, p. 4).

	Input	Output	Content	Expr.	Range	R. Depth	Tasks	Size	Satur.	Verif.
SNLI	NL	MCQA	Common Sense	—	—	—	1	570K	High	✗
MNLI	NL	MCQA	Common Sense	—	—	—	1	433K	High	✗
LogicQA 2.0	NL	MCQA	Common Sense	—	—	—	2	15708	High	✗
Hellaswag	NL	MCQA	Common Sense	—	—	—	1	70K	High	✗
RuleTaker	CNL	Binary	Common Sense	S. Datalog	Low	5	1	500K	High	✗
ProofWriter	CNL	Proof Trees	Common Sense	S. Datalog	Low	5	3	500K	High	✓
ProntoQA	CNL	COT	Common Sense	PHL	Low	5	1	—	High	✓
LogicInference	CNL	COT	Common Sense	PL, FOL ₀	Low	5	5	—	High	✓
LogicAsker	CNL	Binary	Common Sense	PL, FOL ₀	Low	1	1	5450	High	✗
LogicNLI	CNL	MCQA	Common Sense	PL, FOL ₀	Low	5	1	30K	High	✗
FLD	CNL	Proof Trees	Common Sense	PL, FOL ₀	Low	8	1	30K	High	✓
NLSat	CNL	Binary	CommonSense	PL	—	—	1	423K	High	✗
Folio	CNL	MCQA	Common Sense	PL, FOL ₀	Low	7	2	1435	High	✗
LogicBench	CNL	Binary, MCQA	Common Sense	PL, FOL ₀	Low	—	2	15K	High	✗
Multi-LogicEval	CNL	Binary	Common Sense	PL, FOL ₀	Low	5	1	1552	High	✗
Entailment Bank	NL	Entailment Trees	Science	—	—	2.7	1	1840	High	✓
ARC (Challenge Set)	NL	MCQA	Science	—	—	—	1	7787	High	✗
TheoremQA	NL	Literals	Science	—	—	—	1	851	Low	✗
MMLU	NL	MCQA	Hybrid	—	—	—	57	15908	High	✗
MMLU Pro	NL	MCQA, Literals	Hybrid	—	—	—	14	12000	High	✗
BBH	NL	MCQA	Hybrid	—	—	—	23	6511	High	✗
BBEH	NL	MCQA, Literals	Hybrid	—	—	—	23	4520	Low	✗
BBEH Mini	NL	MCQA, Literals	Hybrid	—	—	—	23	460	Low	✗
HLE	NL	MCQA, Literals	Hybrid	—	—	—	23	2500	Low	✗
SuperGPQA	NL	MCQA	Hybrid	—	—	—	1	26529	Medium	✗
MATH	NL	Literals	Math	—	—	—	1	12500	High	✗
GSM8K	NL	Literals	Math	—	—	8	1	8500	High	✗
OlympiadBench	NL	Literals	Math	—	—	—	1	6142	High	✗
OlymMATH	NL	Literals	Math	—	—	—	1	200	Medium	✗
Omni-Math	NL	Literals	Math	—	—	—	1	4428	High	✗
FRONTIERMATH	NL	Literals	Math	—	—	—	1	350	Low	✗
MiniF2F	Logic	Tactic Trees	Math	HOL	—	—	1	488	Medium	✓
PROOFGRID	Logic	Proofs	None	PL, EQL	Med	> 100	4/15	3000	Low	✓

Table 1. Qualitative comparison of 33 reasoning datasets. A long dash (—) means NA (not applicable). **Expr.** represents the expressiveness of the underlying logic (when one exists). **Range** represents the diversity of reasoning mechanisms tested by a benchmark. **R. Depth** is the maximum reasoning depth required. **Satur.** is the benchmark’s saturation level. **Verif.** indicates whether a benchmark programmatically verifies the model’s reasoning. NL: Natural Language; CNL: Controlled Natural Language; COT: Chain of Thought; MCQA: Multiple-Choice Question Answering; PL: Propositional Logic; S. Datalog: Stratified Datalog; PHL: Positive Horn Logic; FOL₀ = propositional logic plus a very small fragment of FOL; EQL: (unconditional) Equational Logic; HOL: Higher-Order Logic.

Nevertheless, even that version is nearing saturation, with o3, Grok-4, DeepSeek-R1, every Gemini model from Gemini-2.5 onward and every GPT model from GPT-5 onward all clustering between 84% and 92% accuracy.

Hybrid benchmarks like BBH and BBEH are marked as MCQA even though they typically include COT-based prompting setups, because the elicited COT traces are not evaluated.

Note that some datasets include structured reasoning artifacts but we have nonetheless marked them as unable to verify a model’s reasoning because the models themselves are not required to produce any such artifacts. This is the case, e.g., for LogicNLI (Tian et al. 2021). Its hypotheses are synthetically generated by starting with premises and then applying a sequence of $n \leq 5$ inference rules until a final conclusion p is obtained. The resulting proof trace $p_1, \dots, p_n = p$ is stored in the dataset as the canonical derivation of p . When a model correctly classifies p as entailed by the starting premises, the authors then check to see if the model also accepts the canonical proof’s intermediate conclusions p_i , a test they dub “proof traceability” (Tian et al. 2021, p. 3741). However, this is mostly a behavioral test of response coherence or consistency rather than proof traceability. Since the model produces no explicit reasoning and its internal process is opaque, agreement on p_i offers no evidence that the model actually followed the canonical derivation path.

Some CNL datasets exercise a relatively large number of inference rules but are still classified as low-diversity, meaning that they test a small range of reasoning mechanisms. This is either because the large majority of the rules are drawn from a single logic (e.g., LogicInference supports 19 inference rules, but 17 of them are for propositional logic and only 2 for predicate logic); or because they represent superficial template variants that don't capture any fundamentally new abilities. It could be argued that inference rules in general are just one type of reasoning mechanism, so rule count alone is a poor indicator of reasoning breadth. In addition, there are several inference idioms, pervasive in everyday mathematical and logical discourse, that cannot be captured by simple inference rules operating on sentences. Chief among them is conditional reasoning, where one makes a provisional hypothesis and proceeds to derive consequences from it and other background assumptions, ultimately discharging it and deducing a conditional. To represent this type of reasoning faithfully, one needs a notion of assumption scope, which simple inference rules do not provide. In principle, hypothetical reasoning could be captured by inference rules that operate on sequents rather than plain sentences, but none of these datasets work with sequents.²³ Even if they did, something important would be lost in translation because sequents lack the hierarchical assumption nesting that is a defining feature of conditional reasoning in the wild.²⁴ Reasoning by contradiction is essentially of the same type. Eigenvariables used in universal generalization ("Consider an arbitrary x such that \dots ") provide yet another example of a structured and scope-based reasoning mechanism that cannot be faithfully captured by simple inference rules. Mental-model theorists (Johnson-Laird 1995) would go further and argue that reasoning is primarily semantic, driven by model-based entailment analysis rather than inference rules. This rings true in mathematics, where a typical step is " p follows from \dots ," not an application of a specific low-level rule like modus tollens. It is up to the reader to verify that there is no model in which \dots are true and p is false. (In the setting of propositional logic, PROOFGRID captures this pervasive idiom with NDL_0 , which does away with all inference rules in favor of semantic entailment; see Section 7.1.)

We have listed the depth of reasoning of GSM8K (Cobbe et al. 2021) as 8 even though, strictly speaking, the dataset does not contain any proof trees or other reasoning artifacts. However, the paper that introduced the benchmark notes that "problems take between 2 and 8 steps to solve, and solutions primarily involve performing a sequence of elementary calculations using basic arithmetic operations to reach the final answer," so we use this quantity as a proxy for reasoning depth. Even though OlympiadBench also has 2334 physics questions, we list its size as 6142 because that's the number of math questions it contains. OlympiadBench contains detailed solutions as well (in natural language), although models are only evaluated on their final answers. We have classified Omni-Math as highly saturated because o3-mini's accuracy on that benchmark already exceeded 80% in 2025 (Ballon et al. 2025, p. 2), without falling below 50% on any categories.

We have listed NL for the input form of the problems of LogicQA 2.0 (Liu et al. 2023), because these were obtained from naturally occurring text, adapted from the Chinese Civil Service Examination. The English version was obtained by machine translation and may inject ambiguities and other issues into a problem (though the authors remark that the quality of the translation is substantially better in version 2.0).

For FLD (Morishita et al. 2023), 30K is the size of each variant of the corpus, but there are multiple "dot variants" of the form $\text{FLD}.k$ for $k = 0, 1, \dots, 5$. Each $\text{FLD}.k$ dataset flips at most one design knob at a time (such as the syntactic complexity of the sentences, the depth of the proof trees, the number

²³ A CNL benchmark based on sequent reasoning would be somewhat at odds with the genre. These benchmarks strive to retain natural language, even if it's at the level of a synthetic veneer, as an implicit gesture toward cognitive plausibility—they aim to simulate, however imperfectly, the surface form of human reasoning. By contrast, formal abstractions like sequents, while useful for rigorous analysis of logic and for proof theory, depart significantly from the style in which humans express their inferences in practice, and so their use would introduce structural artificiality into the benchmark.

²⁴ This is not to say that sequent systems can't represent conditional reasoning at all (clearly they can), only that they do so in a very different style. The situation is analogous to the distinction between structured programming languages and assembly/machine languages. The latter don't have the hierarchical nesting that is needed for natural descriptions of algorithms, but they are Turing-complete nevertheless.

of distractors, and so on), so that controlled ablation experiments can be run to see which aspect of reasoning is more challenging for language models.²⁵ In theory, FLD supports full first-order logic, albeit without equality or function symbols. In practice, the fragment it exercises is far more limited, essentially confined to monadic predicates without quantifier alternations. This reflects a broader trend among CNL benchmarks. Even though these are solidly framed in logic, many end up targeting very weak subsets of predicate logic. This is partly due to the sheer complexity of the bookkeeping infrastructure that is needed for a correct implementation of full-blown proofs in first-order logic. Implementing a sound and complete inference system for full FOL is technically demanding. It requires careful handling of variable scope, alpha-conversion, avoiding variable capture, and enforcing a number of subtle caveats. For instance, the witness variable in an existential elimination must be fresh and have no free occurrences in the conclusion. These provisos are not even mentioned in the formulation of existential instantiation in FLD (Morishita et al. 2023, Fig. B.3(a), p. 15). This raises questions. None of these benchmarks provide any guarantee that the generated proofs are correctly implemented and tested. Soundness rests entirely on a Python codebase, typically built from scratch for the purposes of the benchmark. The risk of error is mitigated by the severe restrictions imposed on the richness of the sentences and proofs that are allowed, but the restrictions also mean that these benchmarks are far from exercising the more challenging—and interesting—parts of predicate logic. Using a preexisting and already-vetted proof-engineering system as an off-the-shelf component to handle the drudgery of proof representation and manipulation has the advantage of minimizing the benchmark’s trusted computing base. The creators of FLD note that they avoid the use of external logic systems in order to maintain greater control over data generation, but such use need not constrain the selection of axioms, the depth of proofs, the size of formulas, or most of the parameters that benchmarks are typically interested in tweaking.

NLSat (Richardson and Sabharwal 2022) is notable in that, like the PL_3 division of PROOFGRID, it attempts to build computationally hard inference problems. But it does so not by drawing from well-known families of complex problems but rather by random sampling near the critical region of the 3-SAT phase transition. The resulting problems pose simple yes/no (**sat** or **unsat**) questions. There are no proofs or other reasoning artifacts, so reasoning depth is impossible to measure. The authors do provide an algorithm-specific measure of complexity, namely the average and median numbers of backtracking conflicts and variable assignments reported by the DPPL(T) algorithm used by Z3 when solving the generated problems; the maximum average and median was 22 and 13, respectively.

LogicBench (Parmar, Mihir et al. 2024) and Multi-LogicEval (Patel et al. 2024) include a number of non-monotonic inference patterns embedded in FOL. Multi-LogicEval does not require any proofs, but their results were obtained with zero-shot COT prompting and they performed a manual analysis of the output COT traces.

FRONTIERMATH has 300 core problems, organized into 3 tiers of difficulty, with an extra 50 in a fourth tier, so we have listed 350 as the total size. The table does not include recent math datasets like NuminaMath (Li et al. 2024) and Big-Math (Albalak et al. 2025), because these are much bigger and primarily intended for training (particularly via RL) rather than benchmarking. We likewise omit logic-based datasets like Rosetta (Baek, Shaan Lee et al. 2025), as they are also mainly intended for training. Rosetta could be used for evaluation as well, but only after one has fine-tuned a model on the training split, because the input format of the problems is a deliberately obfuscated notation (even though the problem structure is propositional), in order to force models to focus on reasoning rather than memorization of surface-syntax patterns. As a result, off-the-shelf models would not fare well on these problems, especially in a zero-shot setting, and the dataset does not provide any scaffolding for ICL prompting. Either way, models are not required to produce proofs or any other reasoning artifacts.

²⁵ Multiple corpus variants with different profiles are common in this line of research (a tradition going back to RuleTaker, which introduced five datasets varying by reasoning depth).

Similar issues arise in datasets like HOL Step (Kaliszyk et al. 2017) and LeanDojo (Yang et al. 2023). These are designed to train models to perform highly specific tasks in tactic-based proof systems like HOL and Lean, such as:

- *Premise selection*: Given a goal g and an available result p , determine whether p will be useful in proving g .
- *Next-tactic selection*: Given the current proof-search state (a notion that is specific to tactic-based theorem provers), encoded as a string, along with a ranked list of premises (generated by a premise-selection component) representing available lemmas, predict the next tactic to apply (a function that takes a sequent as input and produces another sequent as output).

Neither was designed to test a model's general ability to reason with or about proofs, and neither provides prompt scaffolding for ICL-testing of off-the-shelf models. If one wrote prompts from scratch (see footnote 28), one could attempt to use LeanDojo as a test set to be given to an off-the-shelf model, but doing so would mostly gauge the model's ability to generate a Lean tactic that is syntactically valid and semantically applicable to the input proof state. This requires the model to be deeply familiar not just with general Lean syntax and semantics but also with its extensive tactic library; and more importantly, the result would not tell us whether the predicted tactic would eventually lead to success.

In general, tactic-based interactive theorem-proving frameworks are primarily geared towards verification rather than presenting reasoning in the structured and hierarchical style that is characteristic of human-written proofs.²⁶ The main representation of a proof in such a framework is a sequence of system-specific tactic applications. These tactics are usually applied in a backward fashion, decomposing a goal into subgoals until the process reaches axioms or trivially solvable leaves. Although this analytical process does operationalize the *search* for a proof object, it does not directly function as a proof in the usual expository or narrative sense; instead, it's an execution trace that records a proof-search trajectory. This is an important distinction. The tactic trace can convey key aspects of strategy (e.g., which lemmas drive the argument, where cases split, or which invariant is maintained), but the overall structure of the reasoning is encoded implicitly and unevenly. It is dispersed across highly procedural steps that can obscure the high-level organization of the argument, especially for non-expert users of the system, instead of presenting it in a stable, declarative, and hierarchically articulated proof scaffold. Lean's core does ultimately generate a proof term, a fully elaborated term in the Calculus of Inductive Constructions that certifies the correctness of the theorem. However, these proof terms are typically opaque and unwieldy, even for experienced users. They are often closer to compiled machine code than to conventional mathematical argument. In practice, even seasoned Lean users seldom engage with these proof terms directly, instead treating the sequence of tactics itself as the canonical proof. As a result, benchmarks based on such systems are better viewed as evaluations of system-specific verification abilities than as direct tests of declarative, human-interpretable reasoning.²⁷

In addition, because Lean is a complex domain-specific formal language, such benchmarks are accessible mostly to models with substantial prior exposure to it. Barring fine-tuning, meaningful performance is unlikely for models that have not been heavily pretrained on Lean code and proofs, and even then, performance can remain sensitive to nuances in the training distribution.²⁸ This makes

²⁶ That said, broader ecosystems grown around such frameworks increasingly support mixed styles, structured proofs, comments, local lemmas, and higher-level proof organization.

²⁷ Of course, even if tactic sequences aren't structured like natural deduction, they still represent how real users build proofs interactively and hence they do capture useful reasoning artifacts. For users working *within* a system like Lean, tactic sequences are familiar and perfectly functional. But while they are useful operational artifacts, they are not accessible and structurally clear explanations of deductive inference, which is what would be needed for measuring interpretable reasoning.

²⁸ One might point out that a system like Lean is just another formal language, and any formal language can be described in a prompt. After all, this is exactly what was done with NDL in PROOFGRID. In principle, this is a reasonable point. If Lean was a simple language, then in-context learning via prompting could suffice. But Lean is a large and multi-layered system with all the inevitable idiosyncrasies that come with such scale. It has a dependent type theory core (the Calculus of Inductive Constructions); thousands of predefined tactics and lemmas; a rich module system; a highly non-trivial unification and elaboration behavior; and so on. In practice, even advanced models fail to generate correct Lean code unless heavily pretrained on it. Prompting might help to guide or structure a solution, but it can't replace deep preexisting familiarity with Lean.

such benchmarks less well suited for evaluating a model's general reasoning abilities. Additionally, running such benchmarks requires substantial Lean-specific infrastructure, including specialized parsing, proof-checking, and goal-state tracking machinery, often tightly coupled to particular versions of the system. The associated tooling overhead places additional constraints on the accessibility and extensibility of such benchmarks.

Finally, the content of benchmarks like LeanDojo is not abstract logic or untyped symbolic reasoning, but rather interpreted formal mathematics from subjects like number theory. As such, the benchmark is deeply contentful: solving its problems depends not only on fluency in manipulating Lean tactics but also on substantial crystallized mathematical knowledge, including a large number of domain-specific results (as well as notational conventions and standard proof idioms). This positions LeanDojo and similar datasets at the far end of the content axis in our taxonomy, where reasoning is tightly entangled with deep formalized subject matter. This makes them useful for evaluating mathematical assistants or theorem provers in specialized domains within the Lean ecosystem, but less suitable as general-purpose benchmarks for testing fluid or broadly transferable reasoning.

The table also omits datasets like NATURALPROOFS (Welleck et al. 2021), which contains a large number of mathematical proofs (32K) in natural language, mostly culled from ProofWiki. Although it is a useful resource, this dataset presents a significant challenge for the streamlined benchmarking of proof-writing ability, as it is not currently possible to reliably evaluate an arbitrary natural-language proof attempt. The two tasks studied in the paper that introduced this dataset were the retrieval and generation of the *references* that appear in a given proof (definitions, other cited theorems, and so on); such references are stored as part of the dataset and hence candidate solutions to both tasks can be mechanically checked for correctness. A subsequent publication by the same research group did introduce proof-writing tasks, both at the global level of the entire proof and at the finer-grained level of the next proof step (Welleck et al. 2022). However, the issue remains: there is no known sound and complete algorithm for evaluating an arbitrary piece of natural-language text that purports to prove a mathematical result. The authors took a pragmatic approach to this by combining manual evaluation by human experts with the use of computable metrics that can serve as proxies for proof quality, such as lexical similarity with the gold proof, using metrics like GLEU and Token F1, and "knowledge grounding" (overlap of the definitions and results cited by the proof attempt with those of the gold proof). Unfortunately, manual evaluation does not scale, while the heuristics are clearly not sound (as the authors implicitly acknowledge). A mathematical result can have many different proofs, each relying on a somewhat different set of previous results and even a different set of definitions. Nevertheless, the metrics show a strong positive correlation with human judgments of correctness, suggesting they are useful proxies. Accordingly, this work should be understood as an interesting pragmatic evaluation methodology, or a framework that can provide a meaningful but imperfect signal about the ability of models to prove mathematical theorems in natural language.

The 488 data points in MiniF2F are divided between 244 test and 244 validation problems with a stratified random split ensuring that each half has a balanced set of problems from each category and level of difficulty. The table marks the maximum reasoning depth of MiniF2F as NA (—) for several fundamental reasons that distinguish it from benchmarks like RuleTaker or ProofWriter, and our own PROOFGRID. First, MiniF2F provides only problem statements, not machine-checkable proofs. This helps to prevent leakage. Although third-party groups have released separate proofs for certain subsets of the corpus, such as the Lean proof pack for the 14 IMO problems in MiniF2F's test split, which was recently published by Yousefzadeh and Cao (2025), these are not part of the benchmark itself. Without a canonical proof corpus, there are no ground-truth objects that can be used to measure properties like proof depth.

Even if a comprehensive proof corpus were available, reasoning depth is a function of the chosen proof system. MiniF2F is a multi-platform benchmark, with problems formalized in Lean, Metamath, Isabelle, and HOL Light. The structure and length of a proof can vary dramatically between these systems. For example, a proof in Metamath, which is based on a minimal set of axioms and a single

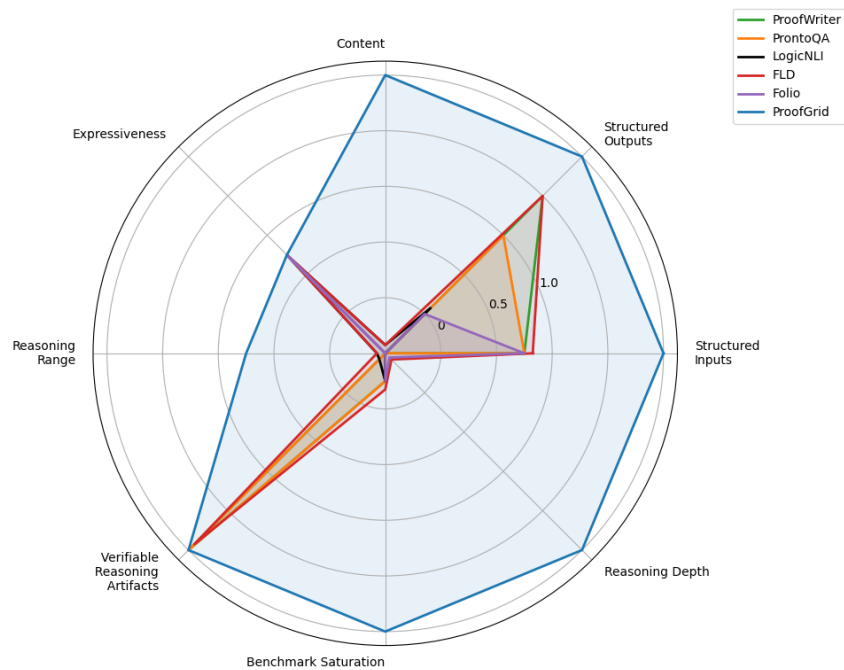


Figure 1. A radar-chart comparison of PROOFGRID and 5 logic-based CNL benchmarks.

rule of inference (modus ponens), is typically a long, linear sequence of low-level steps. In contrast, a proof in a system like Lean or Isabelle can be much more concise by leveraging not only a rich library of previously proven lemmas but also powerful automation tactics. A single-line proof in Lean might correspond to thousands of steps in Metamath, undermining any attempt to define a system-independent depth metric.

Tactics add considerable complexity to the notion of reasoning depth in automated proof assistants. Tactics are high-level abstractions that automate sequences of logical inferences, typically applied backwards (in an analytic and proof-search-flavored rather than a synthetic proof-building fashion). A single call to the `nlinarith` tactic in Lean, for instance, can solve complex non-linear real arithmetic problems by invoking a sophisticated decision procedure that may correspond to many thousands of primitive logical steps. Is this one step or thousands? The answer depends on the desired level of abstraction. This is analogous to measuring program complexity by lines of code: a single call to a powerful library function like `quicksort` represents one “step” at a high level of abstraction, but it encapsulates a complex algorithm with a non-trivial number of primitive operations. Because modern theorem proving relies heavily on such abstractions, a simple “proof tree height” metric fails to capture actual reasoning complexity.

In the case of computation, complexity theory has found a way to talk about the number of steps in an algorithm independently of specific languages or machine models, even when those steps involve arbitrarily complex function calls. This is done with the introduction of key abstractions like Turing machines or RAM programs, which serve as standardized low-level computational models. Even if we write a program in a high-level language, the analysis assumes it can be compiled down to a sequence of basic steps in the low-level machine. So when we say that an algorithm runs in $O(n^2)$ time, we’re measuring how many primitive steps in the underlying computation model are taken, assuming that the high-level constructs are compiled into those. Crucially, this works because the chosen model (e.g., Turing machines or RAM programs) is universal and robust under simulations, owing to Turing completeness: any reasonable programming language can be compiled to it with at most a polynomial overhead. There is no widely accepted analogue for deduction, although in principle one could

fix a low-level proof system that does not allow for tactics or proof search, and then approximate the reasoning depth of a rich high-level proof—one that does use tactics and performs arbitrary proof search—by measuring the size of the “expanded” certificate in the bare-bones formalism.²⁹ Theoretically, this is defensible. It is the approach taken in proof complexity theory, for example, where the analogue of time complexity is proof length, usually defined as the number of symbols or steps in a derivation under some fixed low-level system (such a Frege system, resolution, or cutting planes). Different proof systems are then analyzed on the basis of how efficiently they can prove tautologies (e.g., with some theorems requiring exponentially long proofs in one system but not in another).

However, in practice proofs introduce complications that do not arise—at least not to the same extent—in the case of pure computation. In particular, there is often a sizable conceptual gap between the formal depth of a machine proof and the intuitive difficulty that a human perceives. A problem that a mathematician considers to have a “two-step” solution might require hundreds or thousands of formal steps to rigorously prove from first principles, including the handling of tedious side conditions and corner cases that a human would dismiss as trivial. Conversely, a problem with high conceptual difficulty for a human might have a short, brute-force proof that is easily found by machine. Thus, even though a general and domain-independent notion of formal reasoning depth could be calculated, it would not necessarily correlate with the depth that is relevant to human mathematical reasoning.

In summary, due to the absence of a canonical proof set, the profound dependency on the chosen formal system, the variable granularity of tactics, and the disconnect from human-centric problem difficulty, any single number for the “maximum reasoning depth” of MiniF2F would likely be arbitrary. Similar remarks apply to the range of reasoning mechanisms exercised by MiniF2F, which we have also marked as NA.

The radar chart in Fig. 1 provides a visual comparison of PROOFGRID and 5 reasoning benchmarks along eight evaluation axes that reflect structural, logical, and operational characteristics. Each axis is normalized to a $[0, 1]$ scale, where higher values indicate more favorable properties (e.g., more structured outputs, deeper reasoning, less saturation, and verifiable reasoning artifacts).³⁰ A small amount of jitter is added to the values of some benchmarks to prevent visual overlap in the plot.³¹ PROOFGRID is the only dataset that scores maximally on all axes except expressiveness and reasoning range, where its values are moderate. Overall, the plot illustrates how many CNL benchmarks cluster around low to medium values across most axes, while PROOFGRID occupies a distinctive position as a structurally rich and logically grounded benchmark with a broad and deep inferential footprint.

3. PL₁

3.1. Proof Writing

3.1.1. Data

PL₁-PW contains 400 problems generated by the algorithm described in Section 1. Half of these problems are *conditionalized*, meaning that they contain no premises, while every argument in the other half has at least one premise. Each conditionalized argument α was obtained from an argument that initially had $n > 0$ premises p_1, \dots, p_n and a goal q , by setting α 's goal to the conditional $(p_1 \wedge \dots \wedge p_n \Rightarrow q)$ and removing all premises.

²⁹ Section 1.8.2 of Arkoudas (2001) introduces a rigorous definition of deductive cost in exactly this way.

³⁰ For example, maximum-reasoning-depth values were normalized with min-max scaling, by replacing each value d by

$$\frac{d - d_{\min}}{d_{\max} - d_{\min}}$$

where d_{\min} and d_{\max} are the smallest and largest maximum reasoning depths. For the output axis, we set: Binary:0, MCQA:0.2, Literals:0.4, COT:0.6, Proof Trees/Entailment Trees:0.8, Proofs:1.0. And likewise for the remaining dimensions.

³¹ This occurs when two or more benchmarks have identical scores across all axes—for example, LogicNLI and Folio share nearly the same profile. Without jitter, the polygon representing one benchmark would be completely obscured by the other, making it invisible in the visualization. The jitter values are very small (typically around 0.03) and do not materially affect the interpretive significance of the plot; they only serve as a visual disambiguation aid.

Quantity	Mean	SD	Median	Min	Max	Spark
Number of premises	2.51	0.62	3.0	1	4	
Total problem AST size	28.12	13.76	33	6	69	
# of negations per problem	3.17	2.16	3	0	10	
# of conjunctions per problem	4.12	2.78	4	0	12	
# of disjunctions per problem	3.1	2.02	3	0	8	
# of conditionals per problem	3.84	2.62	4.0	0	14	
# of biconditionals per problem	0.28	0.5	0	0	2	
# of propositional atoms per problem	3.15	1.09	3	2	7	

Table 2. Data distribution summaries with inline box-sparklines for the 400 problems in **PL-PW1**. Box = IQR (Interquartile Range); red line = median; dot = mean; whiskers = 5–95th.

Table 2 summarizes the distributions of several key features of these problems. The first row, number of premises, pertains only to the 200 arguments that are not conditionalized. The second row shows the distribution of problem AST sizes. This is defined, for a given problem, as the sum of the sizes of all premise ASTs plus the size of the goal AST.³² This is not reflected in the table, but the 200 conditionalized problems tend to be larger than the other 200 problems. The next five rows show the distributions of the five logical connectives across all 400 problems. Conjunctions are the most common formulas, followed by conditionals, negations, disjunctions, and finally biconditionals, which occur very infrequently (averaging only 0.28 occurrences per problem).

The last feature, the number of atoms (propositional variables) in a problem, is particularly important. This is a core indicator of logical difficulty. In this task, the median number of atoms is 3, and in fact 75% of all problems have no more than 3 atoms. In general, the number of propositional atoms bounds the logical space sharply, since n atoms admit exactly 2^n possible truth assignments. If $n = 3$, only 8 valuations need to be distinguished, so no matter how complicated the premises might appear syntactically, the underlying semantic structure is small. This is a key reason why **PL1-PW** problems are so tractable—because they collapse into small finite case analyses.

This is not to say that the structure and size of the premises and the goal are immaterial. These can pose syntactic and structural challenges: how deeply formulas are nested, which connectives occur where, how many subproofs must be opened and discharged, and how effectively the relevant introduction and elimination rules can be coordinated. All of these factors can affect proof discovery. However, when the overall number of atoms is small, problems with many or deep premises often have redundancies, contradictions, or tautologies that reduce to something simple once the cases are enumerated. Semantic difficulty, by contrast, correlates strongly with the number of truth assignments that can be distinguished, and this scales directly with the number of atoms. A central challenge in proof discovery, at least in propositional logic, arises from the explosion of the search space that occurs as the number of atoms increases.

3.1.2. Proof Checker Instrumentation

As an example, consider this relatively large conditionalized problem in **PL1-PW**:

³² The size of a formula AST (abstract syntax tree) is simply the number of nodes in the tree.

```
((A ==> (~ (E | ~C))) | (~B & (C | E))) & ((E & ~E) | (B ==> F)) & E & B ==> ((B | C) & (B ==> (F & ~A)))
```

This might look intimidating at first glance, but a more careful inspection of its structure, clarified by writing the formula in prefix notation, shows that it is not difficult to prove:

```
1 (==> (& (| (==> A
2           (~ (| E ~C)))
3           (& ~B
4             (| C E)))
5           (& (| (& E ~E)
6             (==> B F))
7             (& E B)))
8 (& (| B C)
9     (==> B
10      (& F ~A))))
```

Assuming the antecedent, which is a conjunction of several formulas, including E and B , we need to derive the conjunction of $(B \vee C)$ and the conditional $(B \Rightarrow F \wedge \neg A)$. Since we are already given B , deriving $(B \vee C)$ is trivial (by disjunction introduction). For the second part, to derive $(B \Rightarrow F \wedge \neg A)$, we can assume B (which we already have anyway) and show that both F and $\neg A$ hold. We are given that B implies F ,³³ so, since we have B , modus ponens gives F and the only remaining goal is $\neg A$. The latter follows by a *reductio*: if we had A , we could easily derive a contradiction by a case analysis of the antecedent's first conjunct (the disjunction on lines 1–4). This is, essentially, the proof. Translating it into NDL is a bit tedious but not complicated.

The best models get this reasoning exactly right. Gemini-2.5-Pro, for example, offers the following lucid account of its proof strategy:

The overall strategy is to prove the main conditional by assuming its antecedent and deriving its consequent. The antecedent is a large conjunction, which is first deconstructed into its four main components:

1. $((A \Rightarrow (\neg(E \mid \neg C))) \mid ((\neg B) \& (C \mid E)))$,
2. $((E \& \neg E) \mid (B \Rightarrow F))$,
3. E , and
4. B .

The consequent, $((B \mid C) \& (B \Rightarrow (F \& \neg A)))$, is also a conjunction, so its two parts are derived separately. The first part, $(B \mid C)$, follows directly from component (4), B , by disjunction introduction. The second part, $(B \Rightarrow (F \& (\neg A)))$, is proven by a conditional proof, assuming B . Within this subproof, F is derived by a case analysis on component (2). The first case, $(E \& \neg E)$, is a contradiction from which F follows trivially. The second case, $(B \Rightarrow F)$, yields F by modus ponens with the assumed B .

Next, $\neg A$ is derived, also by a case analysis, on component (1). The first case, $(A \Rightarrow (\neg(E \mid \neg C)))$, leads to a contradiction if A is assumed (given component (3) E), thus proving $\neg A$. The second case, $((\neg B) \& (C \mid E))$, contradicts component (4) B , so $\neg A$ again follows from the contradiction. With both F and $\neg A$ derived, their conjunction is formed, completing the conditional proof. Finally, the two parts of the main consequent are joined together to yield the desired conclusion.

Nevertheless, Gemini fails to transcribe this nice piece of reasoning into NDL. Here is (the top part of) its output proof:

```
1 assume h1 := (((A ==> (~ (E | ~C))) | ((~ B) & (C | E))) & ((E & ~E) | (B ==> F))) & E & B {
2   # Assuming the antecedent is left-associative, we decompose it into its four main parts.
3   p1_and_p2_and_E := (((A ==> (~ (E | ~C))) | ((~ B) & (C | E))) & ((E & ~E) | (B ==> F))) & E
4     BY left-and on h1;
5   B_premise := B BY right-and on h1;
6   ...
7 }
```

The proof goes wrong early, on the step shown on lines 3 and 4. In NDL, conjunction associates to the right (as is specified in the prompt), so the hypothesis $h1$, i.e., $(p_1 \& p_2 \& p_3)$, should be understood as $(p_1 \& (p_2 \& p_3))$, not as $((p_1 \& p_2) \& p_3)$. Consequently, the application of `left-and` to $h1$ yields p_1 ,

³³ More precisely, we are given (lines 5 and 6) the disjunction of $(E \wedge \neg E)$ and $(B \Rightarrow F)$, and this disjunction is equivalent to $(\text{false} \vee (B \Rightarrow F))$, which, by $\neg\text{false}$ and disjunctive syllogism, yields the conditional $(B \Rightarrow F)$

not $(p_1 \& p_2)$. Here, p_1 is $((A \implies (\sim(E \mid \sim C))) \mid ((\sim B) \& (C \mid E))) ((E \& \sim E) \mid (B \implies F))$ and p_2 is E , and the conclusion advertised by Gemini on line 3 is $(p_1 \& p_2)$, so the NDL interpreter flags this as an error.

Strictly speaking, this is an error, since the prompt explicitly specifies that conjunction is right-associative in NDL. Yet it is a trivial error in an overall sound proof. Indeed, the model's underlying reasoning is impeccable. One could argue that denying credit to it on account of such peccadillos would be like nitpicking spelling errors in a love letter (though see p. 37 for a different viewpoint). There are many other examples of similar minor failures, running the gamut from syntax to semantics: failing to balance parentheses or curly braces properly, forgetting a semicolon or inserting extra semicolons, using square brackets instead of parentheses, splitting a line of code and putting part of it inside a comment (following a hash sign), failing to distinguish between p and $\neg\neg p$, applying from-false to false itself instead of the desired conclusion,³⁴ confusing the left and right components of a conjunction, disjunction, or biconditional, and many others.

For that reason, we carefully instrumented the NDL proof checker to systematically ignore such errors while preserving soundness. The instrumented checker starts by repairing all syntax errors it can reasonably address. For example, it might insert a missing pair of curly braces, add or delete semicolons, or replace square brackets by parentheses. As an example of a more substantive syntactic intervention, it might change rule applications that use **BY** with rule applications that name their conclusions and do not use **BY**. Models often generate steps like the following (this example is taken from llama-3.1-405b, but all models make this type of error):

$$C\text{-or-not-A-to-not-B } \mathbf{BY} \text{ mp } \mathbf{on} \text{ left, antecedent;} \quad (2)$$

In NDL, only a formula p can appear immediately to the left of **BY**. In particular, a step of the form $p \mathbf{BY} \dots$ imposes a constraint on the inference \dots : its conclusion must be identical to p . It follows that the syntax of (2) is incorrect,³⁵ because $C\text{-or-not-A-to-not-B}$ is not a formula that can be compared against the result of the rule application to the right of **BY**. It's just a name that the model wants to give to the result of that application. Accordingly, our instrumented checker will automatically repair this into:

$$C\text{-or-not-A-to-not-B} := \text{mp } \mathbf{on} \text{ left, antecedent;}$$

After syntactic corrections are made, the checker starts evaluating the repaired proof step-by-step while overlooking a wide class of mostly fiddling semantic and structural errors, such as marginally incorrect applications of inference rules. For instance, the instrumented checker is insensitive to double negation. If a conclusion ought to be p but the proof derives $(\neg\neg p)$ or vice versa, the checker does not complain. The checker is likewise insensitive to conjunction and disjunction commutativity, e.g., it will tolerate a step where an expected conclusion is $(A \wedge B)$ but the model derives $(B \wedge A)$. It will even endorse applications of left-and that should instead be applications of right-and, and vice versa; and likewise for left-either and right-either, as well as left-iff and right-iff. The full list of interventions is extensive, but essentially, if the model makes a logically sound inference but bungles the execution, the instrumented checker will attempt to overlook it and continue with the rest of the proof.

The line is drawn at gross logical errors, such as trying to derive a conclusion that does not follow logically from the current assumption base, or severely ill-formed rule applications, such as applications of conjunction elimination to a disjunction. To give an idea of the scale of these repairs, over the course of evaluating the 400 proofs written by o4-mini, the instrumented checker detected and was able to recover from 649 syntax errors and 363 semantic errors, for an average of 2.53 errors per

³⁴ from-false is a unary inference rule that takes any formula p as its argument and delivers p back as its conclusion, provided that the assumption base contains *false*. This captures the principle that "anything follows from *false*." Thus, e.g., if we wanted to derive $(A \wedge B)$ because the assumption base contains *false*, we would write from-false **on** $(A \& B)$. Instead, models often write vaguely plausible but incorrect variants like $(A \& B) \mathbf{BY}$ from-false **on** false.

³⁵ This is putting it mildly. A step of that form suggests deep conceptual confusion about the role of **BY**.

Model	Overlooked Syntax Errors	Overlooked Semantic Errors	Model	Overlooked Syntax Errors	Overlooked Semantic Errors
hermes-4	2525	245	deepseek-r1-0528	1971	313
llama-3.1-405b	1387	232	deepseek-v3.1	1379	199
deepseek-reasoner	1354	194	gpt-4o	1027	171
gpt-4.1	1077	115	o3-mini	713	316
o4-mini	649	363	mistral-magistral	686	315
deepseek-v3	721	219	o3	318	378
gemini-3.0-pro-preview	520	152	claude-opus-4-20250514	464	182
gemini-2.5-pro	176	291	claude-sonnet-4.5-ext-thinking	224	192
grok-4	177	235	grok-3	187	214
claude-sonnet-4-20250514	205	188	gpt-5	72	274
gpt-5.1	131	156	gpt-5.2	71	8
gemini-3.1-pro-preview	52	12	claude-opus-4-5-20251101	12	29

Table 3. Per-model numbers of syntactic and structural or semantic errors detected and repaired by the instrumented proof checker when evaluating PL1-PW proofs, sorted by the total number of errors.

Model	Instrumented Accuracy	Strict Accuracy	Model	Instrumented Accuracy	Strict Accuracy
claude-opus-4-5-20251101	0.99 (± 0.01)	0.93 (± 0.03)	gemini-3.1-pro-preview	0.99 (± 0.01)	0.92 (± 0.03)
gpt-5.2	0.98 (± 0.01)	0.84 (± 0.04)	gpt-5.1	0.97 (± 0.02)	0.78 (± 0.04)
gemini-2.5-pro	0.94 (± 0.02)	0.60 (± 0.05)	gpt-5	0.92 (± 0.03)	0.62 (± 0.05)
gemini-3.0-pro-preview	0.92 (± 0.03)	0.67 (± 0.05)	claude-sonnet-4.5-ext-thinking	0.88 (± 0.03)	0.63 (± 0.05)
grok-4	0.85 (± 0.03)	0.59 (± 0.05)	o3	0.80 (± 0.04)	0.53 (± 0.05)
deepseek-reasoner	0.76 (± 0.04)	0.45 (± 0.05)	deepseek-r1-0528	0.60 (± 0.05)	0.21 (± 0.04)
claude-opus-4-20250514	0.57 (± 0.05)	0.38 (± 0.05)	o4-mini	0.57 (± 0.05)	0.30 (± 0.04)
deepseek-v3.1	0.49 (± 0.05)	0.26 (± 0.04)	claude-sonnet-4-20250514	0.46 (± 0.05)	0.33 (± 0.05)
o3-mini	0.41 (± 0.05)	0.25 (± 0.04)	grok-3	0.38 (± 0.05)	0.27 (± 0.04)
gpt-4.1	0.35 (± 0.05)	0.23 (± 0.04)	deepseek-v3	0.33 (± 0.05)	0.21 (± 0.04)
hermes-4	0.32 (± 0.05)	0.10 (± 0.03)	mistral-magistral	0.20 (± 0.04)	0.12 (± 0.03)
llama-3.1-405b	0.16 (± 0.04)	0.04 (± 0.02)	gpt-4o	0.15 (± 0.03)	0.10 (± 0.03)

Table 4. PL1-PW proof-writing accuracies from the instrumented proof checker and from the regular (strict) proof checker, with Wilson confidence intervals.

proof. For DeepSeek-R1-0528, the total number of corrections exceeded 2500, averaging 6.93 corrections per proof. Table 3 shows detailed numbers for all models.

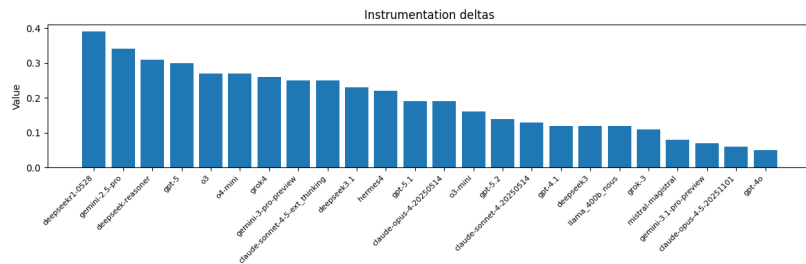
Observe that less able models tend to make many more syntax errors than semantic errors. By contrast, top-tier models tend to make fewer syntax errors, with some of them even making more semantic errors than syntax errors (e.g., o3, Grok-4, GPT-5.1, Gemini-2.5-Pro, and GPT-5). Note also that there is a moderate negative correlation between model accuracy and the total number of error repairs (syntax plus semantics) for a given model: the Pearson correlation coefficient between these two random variables is -0.52. Nevertheless, we will see in the next section that even though stronger models generally receive fewer repairs, they tend to benefit from them more than weaker models. That's because, for stronger models, these repairs are often enough to push a proof over the finish line and make it correct, because the proof only needed minor repairs to begin with. By contrast, proofs by weaker models cannot be salvaged quite as easily because they typically have additional problems that cut deeper. Essentially, strong models tend to fail cosmetically (even if the errors seem flagrant from an instruction-following viewpoint), whereas weaker models tend to fail structurally.

3.1.3. Results

Table 4 reports results from both the strict and the instrumented NDL proof checkers. The gap between the two is striking and demonstrates that naive evaluation by a conventional proof checker would severely underestimate the proof-discovery ability of these models. A relaxed evaluation algorithm that is able to overlook minor errors reveals that proof writing in PL₁ is solved, with the top models achieving essentially ceiling accuracy. Nevertheless, even with the instrumented checker, the task has sharp discriminating power, with performance falling dramatically among mid-tier and lower-end models, as R1-0528, o4-mini, and Claude Opus 4 are unable to reach 70% accuracy, while o3-mini and Claude Sonnet 4 sit well below 50%.

This will be a recurring theme in PROOFGRID: all tasks, even those with strong ceiling effects, are able to tease apart models of different ability levels quite effectively. In Section 3.2 we define and study DI (discriminability index), a simple metric that gauges how well benchmark scores (usually accuracies) manage to separate a set of models S . It is simply the average of score differences between all pairs of distinct models in S . Expressed as a percentage, the DI for the entire set of **PL1-PW** results is 34, a very high value.

Both the average and the median boost in performance brought about by instrumentation is 19 absolute percentage points. This means that, on average, if only strict accuracy was reported, a model's performance on this task would be underestimated by 19 absolute points. The following figure displays the accuracy deltas sorted from biggest to smallest:



The table below provides finer-grained information on the distribution of these deltas:

	Mean	Median	Min	Max	Std	@5	@25	@50	@75	@90	@95
Accuracy Delta	19	19	5	39	9	6	12	19	26	31	34

The benefit of instrumentation follows a roughly inverted-U pattern with respect to baseline accuracy, as shown by the scatterplot of strict accuracy versus instrumentation boost in Fig. 2. We explain this pattern below.

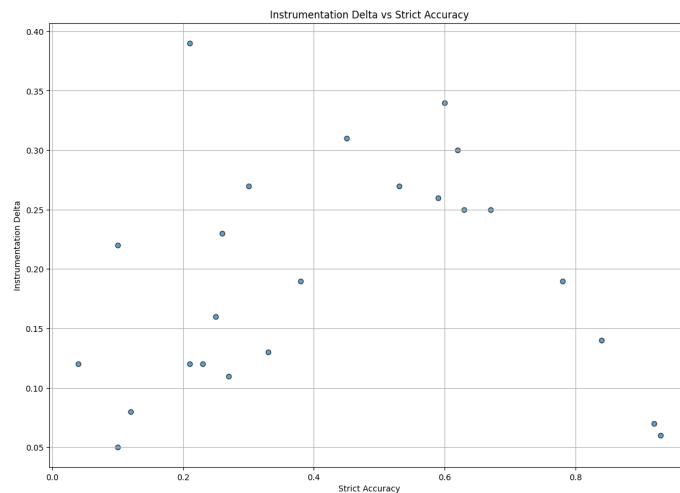


Figure 2. Scatterplot of strict accuracy vs instrumentation deltas across the 24 models.

If one views proof writing as decomposing roughly into two stages, idea discovery (finding the right proof plan) and rigorous execution (turning that plan into a correct derivation), instrumentation only helps with the second stage. The delta is thus minimal for weak models that rarely discover the correct proof strategy, and also small for the few elite models that already often produce fully valid proofs. In particular, as confirmed by the preceding histogram, the smallest improvement (5 percentage points) is realized by GPT-4o, the worst-performing model, while the next smallest improvement (6 points) is realized by Claude Opus 4.5, the best-performing model. The largest gains by far occur for mid-tier models that often reach the correct high-level proof idea but are unable to express it as a

formally valid derivation. These are models that have latent reasoning competence but weak symbolic discipline. The very best models still receive a decent boost, reflecting occasional slips. Instrumentation cleans those up but cannot increase accuracy much because the margins are tight.

While it is tempting to view proof length as a natural axis for analyzing reasoning behavior, raw proof-length distributions are dominated by differences in which problems a model is able to solve. A stack of box plots depicting proof-length distributions for the 24 models for the set of problems that each model is able to solve would naturally invite cross-model comparisons that would nevertheless be statistically indefensible, since the sets in question vary across models and are not matched by difficulty, which means that the resulting distributions are drawn from different populations. It's possible, of course, to control for problem identity, e.g., by restricting attention to the intersection of problems that are correctly solved by a given set of models, but once that is done we find that proof length is not an informative axis of analysis for this task.³⁶ This should not be surprising: the simplicity of these problems naturally constrains verbosity, as proof styles converge and minimal proof lengths become fairly similar. Nevertheless, to give a concrete sense of the intrinsic length scale of proofs in this task, the table below provides some statistics on the distribution of the *shortest* correct-proof lengths for each problem, irrespective of model identity.³⁷ We stress that these are not the shortest or minimal proofs in a formal mathematical sense, only the shortest ones that were observed.

	Mean	Median	Min	Max	Std	@5	@25	@50	@75	@90	@95
Minimal Correct Proof Length	19.78	19.0	1.0	61.0	12.64	1.0	10.0	19.0	30.0	36.0	40.05

The fact that 90% of the problems can be solved with a proof that needs no more than 36 lines further highlights the low complexity of this task.

3.1.4. Error Analysis

What are the errors that cause proofs to fail even when they are evaluated with the instrumented checker? That distribution is shown in Fig. 3. It distinguishes 6 remaining error types:

1. *Parsing/Syntax errors.* Minor syntax transgressions are already fixed by the instrumented checker; what remains are more deviant parsing errors that cannot be easily detected and repaired. The following examples are illustrative:
 - Writing the entire proof as one single continuous line while also interspersing various comments in it, even though, as noted in the prompt, a comment in NDL extends to the end of the line on which it appears. Alternatively, the proof is written as a list of comment lines, where each line starts with a hash and an English sentence (the actual commentary) and is then followed by NDL code, all on the same line.
 - Single or double quotes randomly inserted for no reason.
 - Conditional subproofs with empty bodies, e.g. (example taken from Mistral Magistral):

```
...
assume case2 := ((~ (A ==> ~B)) | (B | A)) {
  # We will break this down further in the proof.
}
...
```

³⁶ For instance, we might consider the set of problems I_K that were correctly solved by all top K models, say for $K = 10$. If we pick a particular model m_r as a reference model (say Claude Opus 4.5, the highest-performing model), then for each $p \in I_K$ we can compute the distribution of log ratios $\log L_m(p)/L_{m_r}(p)$ for every model m in the set of top K models, where $L_m(p)$ is the length of the (correct) proof generated by m for problem p . The boxplots of these distributions would be directly comparable and would tell us how much more verbose or exploratory each model's proofs are by comparison to the reference model. When we do that, we find that proof lengths are roughly similar, with the median log ratio at 0 and more pronounced length differences emerging only at the fringes (after the 90th percentile).

³⁷ Note that every problem was correctly solved by at least one model, so the quantity "the length of the smallest proof for problem p " defines a total function on this dataset and does not introduce any survivorship biases.

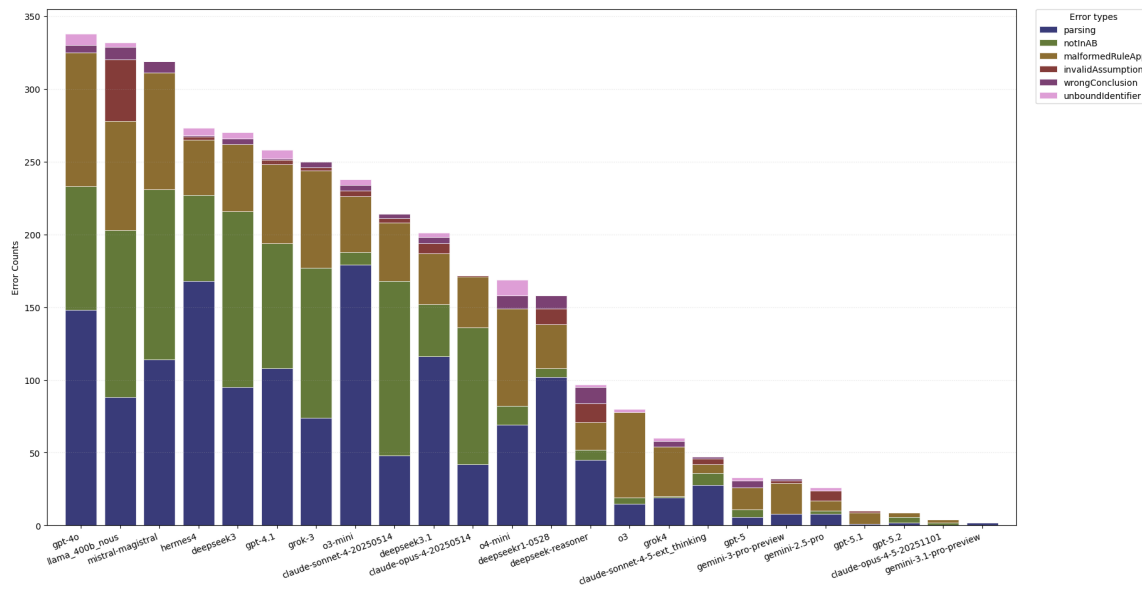


Figure 3. Distribution of first-error types under the instrumented proof checker.

These are like if-then-else statements in Python whose code blocks for the two branches contain only comments and no code. Weaker models probably treat these gaps as placeholders whose details remain undetermined.

- Hallucinating syntax forms that are not part of NDL. Some of these are interesting because they are coherent *generalizations* of NDL syntax. Athena, being a superset of NDL, actually accepts quite a few of these hallucinated forms. For instance, unlike NDL, an argument to a rule application in Athena need not be a simple formula, but can instead be an entire inlined proof, e.g., it is possible to write

```

⋮
C BY cases on (A | B), assume A {⋯⋯}, assume B {⋯⋯};
⋮

```

Here, the second and third arguments to cases are arbitrarily large conditional proofs, presumably deriving the conditionals ($A \Rightarrow C$) and ($B \Rightarrow C$).

Most models wrote at least some proofs in such generalized supersets of NDL. This was no doubt due to transfer learning from the countless instances of syntax compositionality that the models encountered during their pretraining. Because we used Athena as the backbone of our proof checker, we were able to accept and give credit to many such proofs, even though these generalized constructs are not valid according to the prompt.

However, many models push this too far and end up being systematically creative in the wrong direction, by concocting uber-compositional syntax generalizations that are much more extreme and free-ranging than anything allowed in Athena. Here is one example of a single inference rule application written by GPT-4.1 (we added the indentation to clarify the model's intention; the actual output was just one long line):

```

(B & ~A) BY cases on (B | A),
  (B ==> ~A ==> (~A BY mp on (B ==> ~A), B;
    (B & ~A) BY both on B, ~A)),
(A ==> (B ==> ~A),
  (A ==> B),
  A ==> (B BY mp on (A ==> B), A;
    ~A BY mp on (B ==> ~A), B;
    (B & ~A) BY both on B, ~A))

```

While this fails on multiple fronts, the overall intention is relatively clear: The model was trying to write a single application of the cases rule with nested subproofs as the arguments.

In the process, it helped itself to its own bracketing conventions (such as enclosing a proof sequence in parentheses instead of NDL's curly braces) and also embedded the subproofs inside conditional formulas that could not possibly yield the desired conclusions. Here is another example by o3-mini:

```
A_imp_B := (A ==> B) BY assume a := A
      {false2 := false BY absurd on a, h3;
      B1 := B BY from-false on B, false2};
```

These pervasive errors indicate a failure of *cognitive control*, as cognitive scientists might put it, particularly *inhibitory control*—the ability to suppress prepotent or automatic responses, such as the urge to write in natural language or to use familiar compositional syntax patterns from other languages. The models find it difficult to inhibit their pretrained tendencies; they overgeneralize from their training distributions and fail to suppress those generalizations even when explicitly instructed to do so, finding it hard to conform their output only to the constraints explicitly provided in their “working memory” (the context window).

That said, as can be seen from Fig. 3, parsing errors tend to be the minority for most models, particularly for better models. The top performer, Claude Opus 4.5, did not make a single syntax error (the only model to do so).

2. *Not-in-the-assumption-base errors*: These are the most properly logical errors, signifying the application of an inference rule to a formula that is neither a premise, nor a current hypothesis (posited by an **assume** that is still in scope), nor a previously derived conclusion (by a preceding subproof that is still in scope).
3. *Malformed rule applications*: These errors can be viewed as sitting somewhere between syntax and semantics, rather similar to type errors. A typical example is an application of modus tollens to $(B \Rightarrow A)$ and $\neg B$ (an example from Grok-4) or an application of disjunctive syllogism to $(\neg A \vee B)$ and $\neg A$.
4. *Malformed assumptions*: These errors occur in conditional proofs when a model stipulates a hypothesis that is not a valid formula, e.g., **assume** $A_and_B \{ \dots \}$. The model usually intends the malformed assumption as a mnemonic *name* for the actual hypothesis p (in this case, say, the conjunction $(A \& B)$). But instead of writing **assume** $I := p \{ \dots \}$, it writes **assume** $I \{ \dots \}$. (As an analogy from functional programming, this is a bit like writing `let val sqrt_of_n in ... end` instead of writing something like `let val sqrt_of_n = math.sq(n) in ... end`.)
5. *Wrong conclusions*: This is another type of reasoning error, which happens when a model manages to derive a conclusion but not the one that was required. This happens in two cases, either:
 - in a single application of an inference rule where the model uses **BY** to advertise the rule's conclusion as p but the application actually produces $q \neq p$; or
 - at the level of the entire proof, where the model writes a proof that successfully derives *some* conclusion, but not the one that solves the problem.
6. *Unbound identifiers*: This occurs when the proof refers to hallucinated names that have not been defined. For example, in a conditionalized argument with zero premises, the model might still write `premise-1` in reference to, e.g., the first conjunct of the target conditional's antecedent.

These 6 failure modes are mutually exclusive only because they mark *the first error* encountered by the instrumented checker, analogous to a compiler's first error report. It's conceivable that even if that error were somehow disregarded and evaluation continued, the proof would make a second error of a different type (and potentially a third one, and so on). This is like a program whose first compiler error is a syntax error, which is followed by a type error and then an undeclared variable. In general, failed proofs may contain multiple errors, and indeed we have anecdotally verified that many proofs, especially from weaker models, do make both syntactic and logical mistakes. Nevertheless, the “first-error” view afforded by Fig. 3 is informative, because it identifies the earliest bottleneck that would need to be addressed to allow a proof attempt to progress further.

The data shows that the first errors of stronger models tend to be more properly logical or semantic. And although the absolute number of such failures is too small to support fine-grained statistical analysis, this clearly suggests a developmental progression, whereby models first learn to imitate proof structure, then they learn to obey syntax and formatting constraints, and only later become limited by more abstract reasoning challenges.

3.1.5. Discussion

How should we understand the **PL1-PW** results? This task requires two skills:

1. The model must possess a high-level understanding of deductive reasoning and a capacity for strategic-level planning that will enable it to construct an abstract argument, using the inferential machinery of natural deduction, demonstrating how and why the conclusion follows from the premises.
2. The model must also be able to render that abstract argument into a rigorous formal proof.

The results show clearly that models are considerably better at the first task than the second. They have largely sound “mental models” of logical reasoning that they can leverage to solve the straightforward problems in **PL1-PW**. However, they have severe deficiencies in execution—in the *procedural skill* needed to express their reasoning in the precise syntax of a formal language. It is likely possible to mitigate these deficiencies, e.g., by fine-tuning on a target formal language and/or by taking an agentic approach, but the interesting question here is not what could be done with additional engineering effort, but what these results tell us about the native abilities of these models as general-purpose off-the-shelf intelligent systems.

On one view, the procedural difficulties are not alarming. After all, these models had virtually no exposure to NDL during their pretraining, so mistakes are to be expected. These are minor performance errors, not competence errors. Humans, too, when they first see a new language, need a certain amount of practice before they become fluent in it. Even a brilliant logician would almost certainly make some errors on their first attempts at writing in a new formal language.

On the other hand, NDL is an extremely simple language that is fully described in a few pages. Moreover, the task was described in a few-shot setting, with several examples included in the prompt. If a model cannot respect a small number of clearly defined syntax rules, even when illustrated with multiple examples, that by itself arguably indicates a sort of failure of general reasoning, even if it is not, strictly speaking, a failure of logical inference. And while the models might have never seen this *particular* language before, the language is quite standard and very similar to many others that the models *have* seen during their pretraining, in enormous quantities. For instance, NDL uses curly braces { · · · } to delineate scope, semicolons to compose different steps, parenthesized infix syntax for propositional-logic formulas, and so on. None of this is alien to modern models.

Cognitive science gives us a framework for thinking about this: the concepts of near-transfer and far-transfer of learning (Barnett and Ceci 2002). Transfer of learning in general is the application of knowledge or skills learned in one context to a new context.³⁸ Near transfer occurs when we apply knowledge to a new situation that is very similar to what we have seen before (a person who has only driven a Toyota Camry before can almost certainly drive a Honda Accord without any instruction). Far transfer occurs when we apply knowledge to a new situation that is structurally different and seemingly unrelated, say, using principles of chess to inform a business merger strategy. (This is notoriously difficult for humans and is considered a hallmark of deep, flexible intelligence.) Many of the errors made by these models suggest near-transfer failures. Inserting multiple semicolons in a row, writing code to the right of a comment marker and on the same line, or stopping a formula in midstream with unbalanced parentheses are not trivial slip-ups. They indicate that the models are

³⁸ “Transfer learning” (Wang and Chen 2023) has also been a major research paradigm and driver of progress in deep learning, by enabling models trained on one task to perform well on others with limited additional data. While the two traditions overlap, AI has focused on data-efficient reuse of representations, whereas cognitive science is more concerned with the extent to which structural insights can leap across different domains.

failing to apply core known principles to the current specific case and are not attending to the stated rules.

Finally, as we have already stressed, all our tasks are intended as tests of fluid reasoning. We are deliberately using a novel language to prevent the models from relying on crystallized knowledge. The very first step in any fluid-intelligence task is to understand the “rules of the game.” If a system cannot reliably internalize and apply the given rules, it has failed at the most basic level of such a test. Logical deduction is a subsequent step that depends on the first.

In conclusion, while several models at present exhibit a significant capacity for the abstract logical planning required by this relatively straightforward task (as shown by post-correction accuracy rates near ceiling), most of them show a deficit in cognitive control and procedural grounding, failing to autonomously generate fully valid formal proofs, given that after the top 3 models, pre-correction accuracies dip below 80%. Even the very best models currently show a gap between abstract logical competence and rigorous execution. This suggests that the reasoning abilities of language models are not yet fully integrated, instead requiring external “scaffolds” to translate their abstract plans into formally correct proofs. Further evidence for this will appear in subsequent sections.

We close with a few additional observations:

- The instrumented checker catches a very large number of issues, but it is not an oracle. It’s still possible that a proof might fail due to a relatively innocuous error, so even the instrumented accuracies might underestimate abstract proof-writing competence in **PL1-PW**.³⁹ That said, it should not be assumed that models always come up with the right high-level proof plan and fail only in execution. The reasoning strategies themselves, expressed in English, are sometimes misguided.⁴⁰ As an example, consider this problem:

```
assert premise-1 := ((B <=> C) ==> (C & (B | ~A)))
assert premise-2 := (B ==> A)
assert premise-3 := (A ==> (~ (B ==> ~C)))
assert premise-4 := ((A ==> B) & (~C | A))
# Derive: (A & C)
```

A human with some experience would realize that the key is to derive A , which can be done by contradiction. If we had $\neg A$ then modus tollens on premise-2 would yield $\neg B$. Moreover, disjunctive syllogism on the right conjunct of premise-4 would give $\neg C$. Since we have $\neg B$ and $\neg C$, we have $(B \Leftrightarrow C)$ (by the definition of a biconditional), which, by premise-1, leads to C —a contradiction, since we’ve already derived $\neg C$. Now that we have A , C follows easily from premise-3: modus ponens on that and A gives $\neg(B \Rightarrow \neg C)$, which, by the definition of a negated conditional, gives $\neg\neg C$, and hence C .

Here is how GPT-5 describes its reasoning for this problem:

We first extract $(A \Rightarrow B)$ from premise-4 and use premise-2 $(B \Rightarrow A)$. Using these, we prove $(B \Rightarrow C)$ by: assume B , derive A via $(B \Rightarrow A)$, get $\sim (B \Rightarrow \sim C)$ from premise-3, then under a temporary assumption $\sim C$ build $(B \Rightarrow \sim C)$, reaching a contradiction and hence C . This yields $(B \Rightarrow C)$. From A we then derive C by chaining $A \Rightarrow B$ and $B \Rightarrow C$, thus $A \Rightarrow C$. Next, from A we easily get $A \ \& \ C$. To obtain A outright, we appeal to the disjunction $(\sim C | A)$ (from premise-4) and set up a cases argument to yield A ; applying $A \Rightarrow C$ gives C , and finally we conjoin A and C to derive $(A \ \& \ C)$.

³⁹ As a concrete example (taken from deepseek-reasoner), consider a case analysis over a disjunction $(p_1 \vee p_2)$ intended to derive a conclusion q . The model shows that p_1 implies q , but for the second case it shows instead that p_2 implies *false*; it then purports to derive q by applying cases to the 3 arguments $(p_1 \vee p_2)$, $(p_1 \Rightarrow q)$, and $(p_2 \Rightarrow \text{false})$. This application is ill-formed because the two conditionals don’t share the same consequent, so the current implementation rejects it. One could argue that this is a “good enough” case analysis, since $(p_2 \Rightarrow \text{false})$ could readily be transformed to $(p_2 \Rightarrow q)$. But these are the sort of idiosyncratic (and relatively uncommon) errors whose automated detection would need much more sophisticated intent inference and whose repair would cross the line from proof checking to proof synthesis. The current instrumented checker focuses on very common patterns of minor errors that can be easily identified and overlooked.

⁴⁰ Quantifying this would require a rigorous analysis of every proof plan, which would involve manual inspection (since the plans are unstructured text) or the use of other models as judges; we leave this to future work.

This strategy places the emphasis on $(A \Rightarrow C)$ and treats the derivation of A itself as an afterthought, when the opposite is required: Deriving $(A \Rightarrow C)$ is straightforward (though GPT-5 does it via a circuitous route); it's the derivation of A that is (relatively) more demanding. GPT-5's plan erroneously claims that "To obtain A outright, we appeal to the disjunction $(\sim C \mid A)$ (from premise-4) and set up a cases argument to yield A ." But A does not follow from $(\sim C \mid A)$, especially from a simple case analysis of it. The concrete proof written by GPT-5 has no procedural flaw; it follows the plan faithfully:

```

1 ...
2     ## Show that  $(\sim C)$  leads to contradiction with  $A$ -implies- $C$  and  $A$  from the disjunction
3     A_from_disj :=
4     assume  $(\sim C)$  {
5         ## From  $(\sim C \mid A)$  and the assumption  $(\sim C)$ , introduce  $A$  by cases:
6         ## Build the two conditionals needed for cases on  $(\sim C \mid A)$  to reach  $A$ 
7         left_case := assume  $(\sim C)$  { claim on  $A$  };
8         right_case := assume  $A$  { claim on  $A$  };
9         cases on disj, left_case, right_case      ##  $A$ 
10    };
11 ...

```

The problem is that the plan itself is flawed, and so the proof checker rightly reports an error on line 7, complaining that A is not in the assumption base.

- The more common problem with proof strategies is not the presence of outright logical errors but rather the presence of gaps that leave crucial details unspecified. Even the best models do this to a certain degree, but mid-tier and third-tier models often take it to an extreme, with strategies that are so broad and underspecified as to be vacuous. Here is an example from deepseek-v3:

The proof strategy involves deriving a contradiction from the given premises to conclude the disjunction of negations of various propositions. This is achieved by systematically applying inference rules to the premises to derive intermediate conclusions that eventually lead to a contradiction, thereby allowing us to conclude the desired disjunction.

This could be shortened to "The strategy is proof by contradiction."

- Some models use the proof text itself as a thought scratchpad for exploring the proof search space. Mistral Magistral is particularly fond of doing this, as illustrated in the following fragment:

```

1 ...
2     assume B {
3         # If B is true, we must ensure consistency with simplified-premise-2a
4         # Since simplified-premise-2a is  $(\sim (B \mid A) \mid A)$ , and we have  $(\sim A)$  (from our assumption),
5         # then  $(\sim (B \mid A))$  must be false. So,  $(B \mid A)$  must be true. But since we are assuming B, and we
6         # have
7         #  $(\sim A)$ ,  $(B \mid A)$  is true because B is true. However, we need to derive C. Since B is true,
8         # and we have  $(C \mid B)$ , we can derive C or B. But we need C. We need to ensure that C is not
9         # false.
10        # Since we have  $(\sim A)$  and  $(C \mid B)$ , and B is true, but we need C. But  $(C \mid B)$  with B being true
11        # doesn't help us get C. So, perhaps another approach is needed. Let's try to derive
12        # a contradiction if C is false.
13        assume  $(\sim C)$  {
14            false BY absurd on C,  $(\sim C)$  # But C is not directly available here
15        };
16        # Instead, let's use premise-3 to derive  $(C \mid B)$ , which we already have.
17        # But since we have B (from the assumption), and  $(C \mid B)$ , we can't directly get C unless we
18        # assume  $(\sim B)$ , but we have B. This suggests a dead end, so perhaps we need to reconsider
19        # our approach.
20    }
21 ...

```

Magistral repeatedly makes incorrect inferences and recognizes them as such (see the comment on line 12), but continues unfazed to explore many alternative reasoning paths. The model is clearly performing a search, but it fails to separate its internal monologue and search process from the final polished output. This is another failure of cognitive control.⁴¹

⁴¹ Mistral Magistral does this more than other models, but it is not alone. More than 10% of the proofs written by Claude Opus 4 include comments with phrases like *Wait; Hmm; This is getting too complex, let me try yet another approach; Let me restart with a*

In conclusion, despite the very impressive progress reflected in these results and the near-ceiling performance of a few models (aided by instrumentation), this simple task suggests that models still operate in a semantic soup of sorts. They have learned that the purpose of left-and and right-and is to “get a piece of a conjunction” but they have not mastered the precise operational semantics of how those tools work. They treat them like interchangeable magic wands for pulling things out of a conjunction, rather than as specific, deterministic functions with preconditions and postconditions. This is an abstraction failure. The models have correctly abstracted the goal of the rules but failed to abstract their mechanics. In the semantic soup in which these models operate, a conjunction logically entails each conjunct. For the models, this semantic entailment is the most important signal. The precise syntactic path required to demonstrate that entailment is a secondary, lower-importance detail that they often get wrong. Here is a typical example, from the deepseek-r1-0528 proof for problem 325, which contains the following step: C **BY** left-and **on** (A & C). Even without the precise specification of left-and given in the prompt, it should be apparent to the model that C is the right component of (A & C), not the left one. Even the best models at this time continue to be susceptible to these errors. Here’s an example from GPT-5.2 on an extremely simple problem (item 119) that seeks to derive (A & B) from (B \Leftrightarrow A) (given as premise-1) and B (premise-2):

```
{
  ## From the biconditional, get the direction B ==> A
  B-implies-A := right-iff on premise-1;
  ...
}
```

But right-iff detaches the conditional in the opposite direction, as explained in the prompt (which also contains examples of right-iff in action):

Rule 12: Name: right-iff (biconditional elimination)

Description: Applied to a sentence of the form (p \Leftrightarrow q) that holds, it produces the conclusion (q \Rightarrow p).

Our instrumented proof checker largely validates the models’ semantic reasoning while forgiving their procedural sloppiness (and overlooks this particular error), but mathematics and logic demand precision.

3.2. Proof Checking

Dataset and Task Formulation

PL1-PC contains proofs for 300 PL₁ arguments, only 26 of which are conditionalized. The core task here is to examine each of these proofs and determine if it is correct. If the model deems a proof incorrect, it must also report the first error made by the proof (“first” while reading the proof from top to bottom and from left to right).

38 of the proofs were generated by a theorem prover written in Athena, while the rest were produced by Gemini-2.5-Pro, o3, GPT-4o, and Claude Sonnet 3.5.⁴² The proofs written by the theorem

clearer strategy; and so on. Claude Sonnet does this as well (*This won't work, let me reconsider; Wait, that's wrong. Let me fix this.; Wait, that's not right either. let me reconsider the biconditional; etc.*), as do GPT-4.1 (*... but that's not helpful. instead, let's use the other disjunct of premise3*) and others, occasionally even Grok-4. The only exceptions are the OpenAI models starting with GPT-5 and above, the o-series models, and the Gemini models.

⁴² LLM-written proofs were included in order to broaden the dataset beyond a narrow range of synthetically generated proofs and reasoning errors. A potential concern is that asking models to check proofs that they wrote themselves might introduce an evaluation bias. However, **PL1-PC** evaluates 24 models while the model-generated proofs included in this task were written by only 3 of them (one of the four, Claude 3.5, is not among the models we evaluated). This means that for the large majority of evaluated models (21/24), the “LLM proofs” are never their own outputs. Even for the contributing models, the risk of direct self-evaluation is heavily diluted by the mix with synthetically generated proofs and proofs written by other models. Thus, even in principle, cross-model evaluation ensures that there is no systematic “grading your own homework” effect. We also see no empirical evidence of self-evaluation bias. For instance, Gemini-2.5-Pro performs nearly identically on its own proofs (0.91 accuracy) and on o3-written proofs (0.89), while o3 does worse on its own proofs (0.81) than on Gemini-written proofs (0.92). That said, accuracy deltas might not be the most informative indicator of a self-evaluation bias, so we also computed Δ FPR, that is, the change in false-positive rate when an evaluator judges its own incorrect proofs versus those of others (Δ FPR = $P(\text{accept} \mid \text{incorrect, own}) - P(\text{accept} \mid \text{incorrect, other})$). This targets the specific leniency mechanism

prover were correct by construction, so synthetic errors were randomly injected in some of them. The model-written proofs were not corrupted because they contain organic errors of their own (which is why they were included). The dataset is balanced, with an even split between correct and incorrect proofs. Correctness judgments and ground truth in general come from Athena but undergo extra scrutiny to account for NDL's simplicity, as explained below. The following table presents summary statistics for the distribution of proof lengths across these 300 samples.

	Mean	Median	Min	Max	Std	@5	@25	@50	@75	@90	@95
Proof length	30.69	29.0	5	125	13.85	12.0	21.0	29.0	38.0	49.0	55.05

As the numbers show, these proofs are generally short, with 90% having fewer than 50 steps. Here is an example of a correct proof:

```

1 assert premise-1 := (~B | (~A ==> C))
2 assert premise-2 := (C ==> A & ~B)
3 assert premise-3 := B
4
5 # Goal: A
6
7 {
8   # We proceed by contradiction to derive A. We will assume ~A and derive false.
9   assume (~A) {
10    # We'll use premise-1 to perform a case analysis.
11    case1 := assume (~B) {
12      false BY absurd on premise-3, (~B)
13    };
14    case2 := assume (~A ==> C) {
15      C BY mp on (~A ==> C), (~A);
16      (A & ~B) BY mp on premise-2, C;
17      A BY left-and on (A & ~B);
18      false BY absurd on (~A), A
19    };
20    false BY cases on premise-1, case1, case2
21  };
22  A BY by-contradiction on A, (~A ==> false)
23 }

```

The input prompt consists of a detailed description of the proof-checking task and of NDL, followed by a particular NDL argument (zero or more premises, labeled *premise-1*, *premise-2*, ..., plus the target conclusion) along with an ostensible proof for that argument. The model is instructed to output a JSON dictionary with two keys, *correct* and *errorDetails*. If the proof is correct, the value of *errorDetails* should be the empty dictionary, {}. If the proof is incorrect, then *errorDetails* should have 4 keys whose values collectively specify the first (topmost) error found in the proof:

1. *offendingLine*: the line of the proof containing the error (if such a line exists⁴³).
2. *offendingLineNumber*: the number of the offending line (needed for disambiguation purposes, since the same line might have multiple occurrences in the proof), where counting starts at 1. If there is no specific offending line, *offendingLineNumber* should be set to 0.
3. *errorType*: one of three values: *syntax*, *type*, or *logic*.
4. *errorDescription*: a detailed description of the error.

Regarding the three error types:

that accuracy can mask due to prevalence effects. For Gemini-2.5-Pro vs o3 we find $\Delta \text{FPR} = -0.10$ (no leniency; $p \approx 0.65$), and for o3 vs Gemini-2.5-Pro we find $\Delta \text{FPR} = +0.33$ (suggestive in magnitude but not statistically significant; $p \approx 0.066$ with small n). Taken together, these results, alongside the accuracy comparisons above, provide no compelling evidence of self-evaluation bias in this setting. The real risk of including model-written proofs is the unpredictability of their structure. Synthetic proofs allow fine-grained control over which proof constructs are used, and this greatly simplifies evaluation. Model-written proofs, by contrast, often rely on creative corner-case features, resulting in proofs that are technically invalid under the prompt's constraints yet arguably correct in spirit. Accommodating such cases typically requires painstaking modifications of the evaluation pipeline and/or prompt revisions.

⁴³ For an example where no such line exists, consider a proof that goes through to completion but derives a conclusion other than the expected target. That's an error, but it is not caused by any specific line in the proof.

1. A syntax error can be relatively minor (e.g., an extra or missing semicolon, an unclosed curly brace, an unparenthesized formula, a misspelled rule name, and so on), or something more substantive, e.g., a stretch of text that doesn't parse under the described grammar rules of NDL and whose intended meaning is unclear. The models are instructed to disregard minor syntactic errors.
2. A type error occurs when an inference rule is applied to the wrong number of arguments or to arguments of the wrong form, e.g., an application of modus ponens to 3 arguments or negation elimination to a conditional.
3. Logic errors occur when an inference rule is applied to formulas that do not currently hold, or when a proof step derives a conclusion other than the one specified in the corresponding proof line.

The following is the algorithm for evaluating a model's output:

- If the model's verdict is positive, award it a point as long as the instrumented checker finds the proof correct modulo syntax repairs. No point is given if the instrumented checker finds the proof incorrect or if it performs structural/semantic repairs.
- If the model's verdict is negative, award it a point as long as:
 1. the instrumented checker finds the proof incorrect and reports the same error as the model; or
 2. the instrumented checker finds the proof correct but only after performing structural/semantic repairs, and the first such error it repairs coincides with the one reported by the model.

No point is given if the instrumented checker finds the proof correct without performing any structural/semantic repairs, or if the first such error it repairs is different from the one reported by the model.

Results

Results appear in Table 5. We show accuracies for the full task, as determined by the foregoing method, as well as accuracies for the simplified binary version of the task (is this proof correct, yes or no?). This lets us quantify how much of a model's success comes from understanding versus guessing.

Model	Full Task	Guessing Only	Model	Full Task	Guessing Only
gemini-3.1-pro-preview	0.94 (± 0.03)	0.97 (± 0.02)	gpt-5.2	0.92 (± 0.03)	0.95 (± 0.03)
claude-opus-4-5-20251101	0.92 (± 0.03)	0.94 (± 0.02)	gpt-5.1	0.91 (± 0.03)	0.95 (± 0.03)
gemini-2.5-pro	0.90 (± 0.03)	0.94 (± 0.03)	gemini-3.0-pro-preview	0.89 (± 0.04)	0.96 (± 0.02)
claude-sonnet-4.5-ext-thinking	0.89 (± 0.04)	0.94 (± 0.03)	gpt-5	0.86 (± 0.04)	0.92 (± 0.03)
o3	0.82 (± 0.04)	0.92 (± 0.03)	grok-4	0.82 (± 0.04)	0.88 (± 0.04)
o4-mini	0.79 (± 0.05)	0.87 (± 0.04)	deepseek-reasoner	0.73 (± 0.05)	0.83 (± 0.04)
deepseek-r1-0528	0.73 (± 0.05)	0.82 (± 0.04)	o3-mini	0.68 (± 0.05)	0.79 (± 0.05)
claude-opus-4-20250514	0.65 (± 0.05)	0.79 (± 0.05)	llama-3.1-405b	0.60 (± 0.06)	0.69 (± 0.05)
mistral-magistral	0.49 (± 0.06)	0.63 (± 0.05)	gpt-4.1	0.47 (± 0.06)	0.64 (± 0.05)
deepseek-v3.1	0.45 (± 0.06)	0.60 (± 0.06)	hermes-4	0.42 (± 0.06)	0.63 (± 0.05)
claude-sonnet-4-20250514	0.42 (± 0.06)	0.57 (± 0.06)	grok-3	0.34 (± 0.05)	0.55 (± 0.06)
deepseek-v3	0.23 (± 0.05)	0.46 (± 0.06)	gpt-4o	0.23 (± 0.05)	0.51 (± 0.06)

Table 5. PL1-PC accuracies for the full task and for simple binary classification of proofs as correct/incorrect.

We see that the top 7 models achieve very high accuracies on the full task and nearly perfect accuracies on the simplified ("guessing only") version. Unsurprisingly, when the task is simplified to giving a plain yes/no answer, the biggest beneficiaries are the worst-performing models. We see the accuracy of GPT-4o more than doubling, jumping from 0.23 to 0.51. Huge gains are also seen by DeepSeek-V3, Grok-3, DeepSeek-V3.1, and Hermes 4.⁴⁴ But even very strong models benefit from

⁴⁴ Note, however, that even in this setting these models perform roughly on par with random guessing.

having to simply guess instead of having to explain themselves (e.g., o3's performance jumps up by 10 absolute points), underscoring the point that the surface-level classification required by binary or multiple-choice questions is too easy and severely overestimates model ability.

The deltas are much smaller for the top 5-6 models that already perform very well on the full task, partly because these are already starting to bump up against the ceiling and don't have much room for improvement. While there is indeed top-end compression and a moderate ceiling effect among the elite models, which are clustered within a few percentage points of one another, performance on the full task shows wide variance across the entire set of models, with accuracies gently declining from the low 90s to the low 20s, as can be seen in Fig. 4, showing that even this simple task has significant discriminating power. In particular, the mean accuracy is 0.65 and the standard deviation is 0.23.

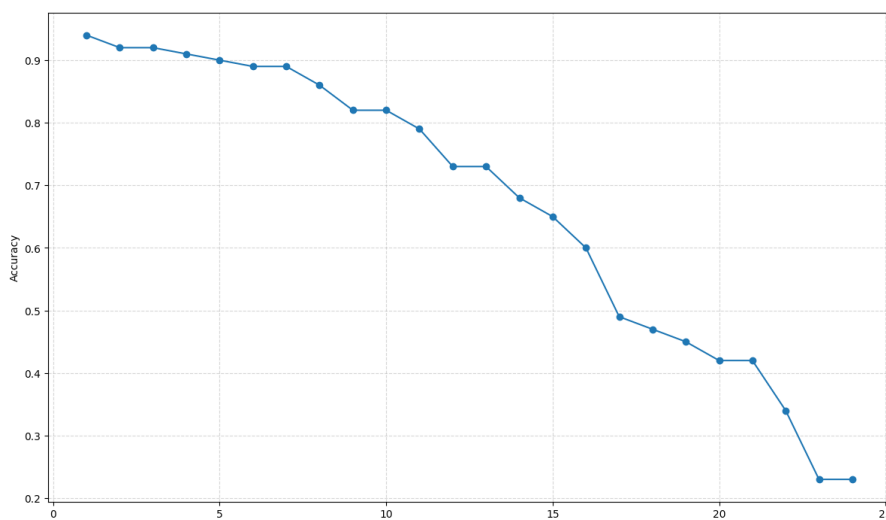


Figure 4. PL1-PC full-task accuracy vs rank.

Quantifying Benchmark Discrimination

Appendix B goes beyond simple variance or pairwise-difference analysis to derive a more theoretically grounded estimate of the discrimination power of this and other PROOFGRID tasks through IRT (Item Response Theory). It also includes detailed *item-level information* for several of the benchmark's tasks. A lighter but still informative alternative to IRT-based discrimination power is a *discriminability index* (DI) that quantifies how well a benchmark separates models. This is a simple measure based on Gini mean difference rather than standard deviation. Intuitively, DI measures how much model accuracies differ from one another. If all models score about the same, DI will be close to 0. If some models are much better than others, DI will be larger.

Specifically, suppose there are n models with accuracies A_1, A_2, \dots, A_n , each between 0 and 1. We look at all possible pairs of models and compute their score differences:

$$DI = \frac{1}{\frac{1}{2}n(n-1)} \sum_{i < j} |A_i - A_j|$$

This is simply the average absolute difference in accuracies between every pair of distinct models.⁴⁵ Because the maximum possible difference between any two accuracies is 1.0, the DI value itself lies between 0 and 1. $DI = 0$ means every model got exactly the same score, so the benchmark cannot tell them apart. DI near 1 means that model performances are spread widely apart and the benchmark is

⁴⁵ As mentioned earlier, this measure, the mean pairwise absolute deviation of model accuracies, is commonly known in statistics as the (sample) *Gini mean difference* (where the random variables are the accuracies of the various models). We refer to it here as a "Discriminability Index" (DI) because in the present context it quantifies how effectively a task differentiates between models.

extremely discriminative. Although DI is theoretically bounded between 0 and 1, such extreme values would require some models to be nearly perfect while others fail entirely. In practice, accuracies across contemporary models occupy a narrower range, leading to DI values typically below 0.3, which we can interpret qualitatively as follows:

0.00–0.05 :	Benchmark strongly saturated (models nearly identical)
0.05–0.10 :	Early signs of compression
0.10–0.20 :	Healthy discrimination
0.20–0.30 or above :	Strong discrimination (large spread)

These categories are not based on strict mathematical criteria; they are practical guidelines reflecting the operating range of DI observed in realistic evaluation settings.

DI naturally depends on the composition of the model set. For instance, restricting attention to a set of models with a narrow range of abilities will yield smaller pairwise differences and thus a lower DI value. Thus, ideally, for rigorous comparisons across benchmarks or tasks, the set of models should be held fixed. But this is a control ideal that may not always be feasible. In practice, when two benchmarks have overlapping and comparably diverse model sets, large differences in observed dispersion remain informative.

In our case, the accuracies of the models evaluated on the full **PL1-PC** task range from about 0.23 (lowest) to 0.94 (highest). Plugging these numbers into the formula above yields $DI \approx 0.27$. This means that, on average, any two models differ by about $|A_i - A_j| = 0.27$ in accuracy, a 27-percentage-point gap. This is a high DI value that lies in the upper region of the strong-discrimination range. It shows that the task clearly separates models of different ability: mid-tier and weaker models score significantly lower than the strongest ones. Only the very best models (those near $A_i \approx 0.9$) are tightly clustered together, hinting at a ceiling effect at the top. But the task still provides meaningful information about model ability, and differences in score genuinely reflect differences in ability.

Finally, unlike standard deviation, which measures distances from the mean, DI directly measures pairwise model differences and is thus more intuitively interpretable. It also tends to be more robust to outliers, and it is in the same family of dispersion measures as the Gini index, which means it connects directly to the geometry of Lorenz curves and inequality coefficients. For instance, Fig. 5 visualizes the information conveyed by the **PL2-PC** DI in a more intuitive way using a Lorenz-style curve. The horizontal axis shows the cumulative share of models, starting from the weakest on the left and moving

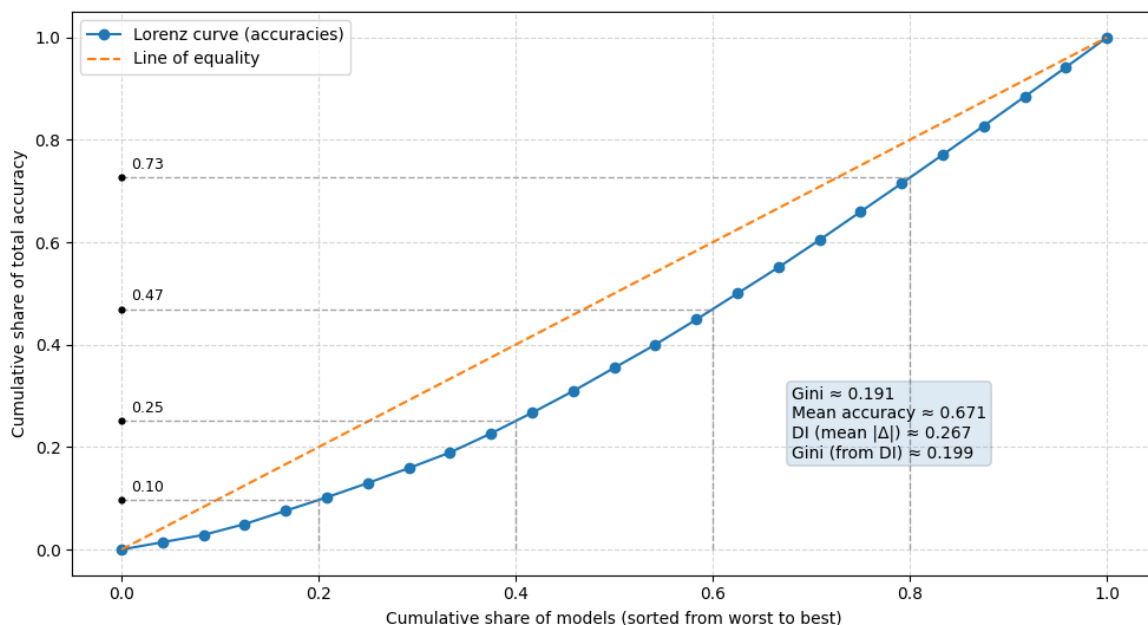


Figure 5. Lorenz-style curve of model accuracies on the full **PL1-PC** task.

to the strongest on the right. Every point along the unit interval on the x -axis represents a fraction of our model population, sorted from the lowest-scoring models on the left to the highest-scoring models on the right. For example, the point $x = 0.4$ represents the bottom 40% of the models in the task, while $x = 1.0$ represents the totality of all 24 models. At $x = 0$, none of the models are included yet. So the x -axis can be thought of as a kind of “percentile rank” axis for the models being tested. It tells us what fraction of that population we have counted so far, starting from the weakest. The vertical axis shows the cumulative share of the total accuracy mass contributed by these models (literally the sum of all models’ accuracies), i.e., $y = 0.23$ represents 23% of the task’s total accuracy mass. In our case the total accuracy mass is 16.1, so the point $y = 0.23$ represents 3.7 units of accuracy. As we move from $y = 0$ to $y = 1$ we consume more and more of that accuracy mass, with $y = 1$ accounting for all accuracy. The dashed diagonal represents a state of perfect equality: if the bottom $x\%$ of models contributed exactly $x\%$ of the total accuracy mass, then the Lorenz curve would coincide with this diagonal. It’s not hard to show that this could only happen if all models have identical accuracies, in which case the task would have zero discriminating power.

Any bowing below the diagonal therefore indicates performance inequality, meaning that some models account for smaller shares of the total accuracy. In our case we see that the bottom 20% of the models contribute only 10% of the total accuracy, while the bottom 40% contribute 25%. By the time we hit 60% we have accounted for 47% of the total accuracy. So we have a modest amount of “inequality” (a Gini index of 0.19⁴⁶) and a gently sloping Lorenz curve, which is precisely what we should want in a benchmark task. Consider three versions of a Lorenz curve for such a task: (a) a straight line along the diagonal, indicating perfect equality and a complete lack of discriminative power; (b) a highly concave curve that is almost flat for most models and only rises sharply towards the end, indicating a state of extreme inequality where only the very best models contribute meaningfully to the total accuracy mass and the task is out of reach for the rest (a kind of model plutocracy); and (c) a moderately concave curve, indicating that stronger models contribute more but weaker ones still contribute significantly. Lorenz curves of the third type are preferable.

Error Analysis

We partition all possible root errors into 5 types:⁴⁷

1. *Type-1 errors*: The model mistakenly claims that the input proof is correct. If we view **PL1-PC** as a debugging task whose objective is the detection of reasoning failures, then type-1 errors can be regarded as false negatives.
2. *Type-2 errors*: The model mistakenly claims that the proof is wrong (false positives).
3. *Type-3 errors*: These represent formatting issues (such as the model’s response missing an expected field).
4. *Type-4 errors*: The model correctly claims that the proof is wrong but fails to pinpoint the first (topmost) error.
5. *Type-5 errors*: The model correctly claims that the proof is wrong *and* manages to pinpoint the first error, but it mischaracterizes the error’s type (syntax/type/logic).

Formatting errors represent instruction-following failures and are rare. The rest of the error types are arranged in order of seriousness: Type-1 errors are perhaps the most serious: failing to detect an

⁴⁶ The Gini index (or “coefficient”) G is the ratio of the area between the diagonal and the Lorenz curve divided by the total area below the diagonal, which is always $1/2$. It measures how far the observed curve lies below the “line of equality.” In general, $G = 0$ means perfect equality and $G = 1$ means maximal inequality (one model has perfect accuracy and all others fail completely). Like DI, G quantifies spread but does so geometrically. In our case, we constructed the Lorenz curve for **PL1-PC** from the task data using interpolation and we computed G by numerical integration (using the trapezoidal rule). The “Gini from DI” value shown in the plot is derived from the theoretical approximation of G as $DI/2\mu$, where μ is the mean accuracy. This becomes an identity in the limit (with larger numbers of models n). Even with only $n = 24$ models in **PL1-PC**, this approximation (0.199) is quite close to the actual value found by integration (0.191), which indicates internal consistency between these two ways of quantifying task discrimination and strengthens the conceptual link between DI (which we introduced for intuition) and the Gini coefficient (borrowed from inequality theory).

⁴⁷ By a “root error” we mean the *first* error that ought to be reported according to the task instructions.

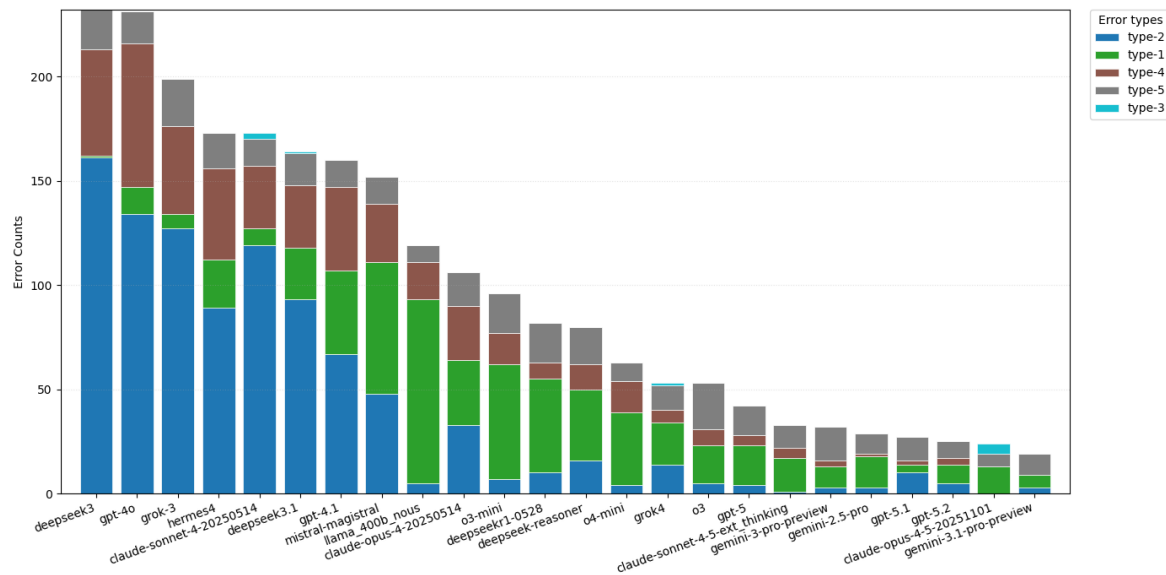


Figure 6. Types of root errors made by models on **PL1-PC**.

incorrect proof. Type-2 errors (mistakenly claiming that a proof is incorrect) are also fundamental. These are the green and blue errors in the plot, and together they make up the majority of errors for most models, especially the weaker ones. Type-4 and type-5 errors can only occur if the model has already correctly classified the proof as wrong. Accordingly, these errors (brown and gray) expose more subtle metacognitive weaknesses: failures in error localization or error characterization. Type-4 errors, in particular, occur when the model fails to identify the root problem in an incorrect proof. Instead, the model singles out another error (since the proof may contain multiple errors) or it hallucinates a spurious one. Type-5 errors are the most innocuous. Here the model properly calls out an incorrect proof and successfully identifies the first error but classifies it incorrectly (e.g., miscategorizing a syntax error as a logic error).

Figure 6 plots the distribution of these error types across the set of all root failures by all models. Among the top-performing models, false negatives (type-1 errors) tend to be more common than false positives (type-2 errors). Gemini 3.1 and Claude Opus 4.5 are the only models to make zero type-4 errors, highlighting their strong debugging skills. That said, in this task the number of errors among the top models is so small that detailed analyses are unlikely to yield statistically robust conclusions.

Among the rest of the models, we see clear behavioral similarities between DeepSeek-V3, Grok-3, and GPT-4o. All three are strongly inclined to find fault with correct proofs, with type-2 errors for these models exceeding all other error types combined. DeepSeek-V3, in particular, rejects 291 out of the 300 proofs, consistent with the overly cautious profile it manifested in other PROOFGRID tasks. This tendency was tapered somewhat in DeepSeek 3.1, which rejects 200 out of the 300 proofs, but that is still a lot of false positives. The situation is largely reversed for llama-3.1-405b, which accepts 84% of all proofs as correct. Mistral Magstral strikes a balance between false positives and false negatives.

Performance vs Input Size

Most proofs in **PL1-PC** are short, so there is insufficient data for a conclusive analysis of proof-checking performance as a function of proof length. Nevertheless, for the better models anyway, and in this particular evaluation regime of short proofs without intricate logical structure, the data suggests that performance is not significantly affected by proof length. This is in contrast to proof writing and other generative reasoning tasks, where performance is more sensitive to input size, even for smaller problems. Fig. 7 plots proof-checking accuracy as a function of proof length for the top 10 models. Proof length stretches to no more than 50, excluding comments and blank lines.

To clarify the relationship between accuracy and proof length, the proofs have been divided into variable-width bins along the horizontal axis. Each bin covers a range of proof lengths (for example,

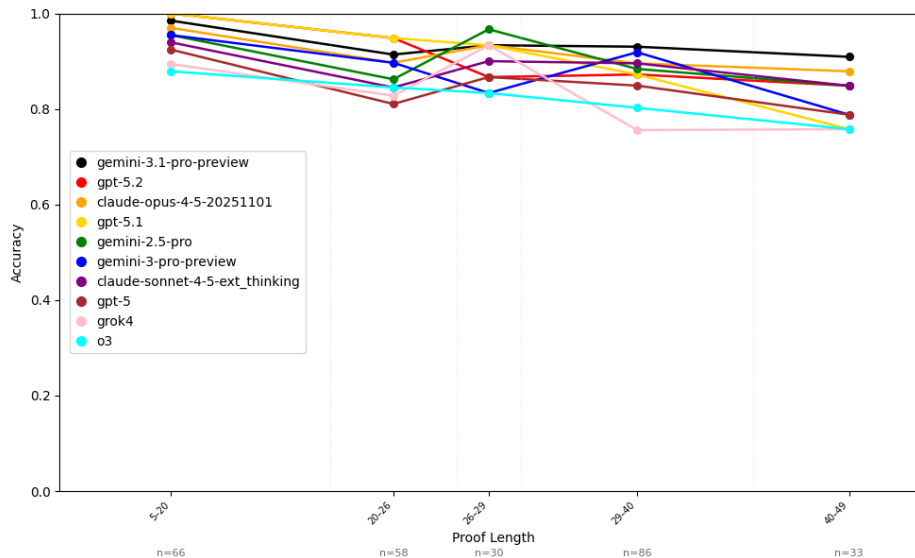


Figure 7. PL1-PC accuracies for the top 10 models as a function of proof length.

5–20, 29–40, and so on), and the points show the average accuracy for all proofs in that bin. The bins are variable-width because the distribution of proof lengths is uneven: there are many short proofs but far fewer long ones. Using equal-width bins (e.g., every 10 units of proof length) would have produced some bins with too few samples to estimate accuracy reliably, particularly at the long-proof end. Instead, we have automatically adjusted the bin widths so that each bin includes roughly a minimum number of n samples (here $n \geq 30$). Extremely sparse tail bins (where proof lengths exceeded 100 and only one or two examples were available) were dropped entirely to avoid misleading fluctuations. Thus, the final range reflects all bins that passed this minimum sample threshold. Beneath each bin label on the x -axis, the number of proofs (n) used in that bin is printed, showing the data density across the proof-length spectrum. This adaptive binning, which we use for all similar plots in this paper, gives a clearer and fairer picture of performance across the range, instead of overemphasizing regions with plenty of data or misrepresenting sparse regions.

All of these models start out strongly and maintain high performance throughout, experiencing only gentle declines as proof length increases. The best of them maintain accuracy strictly above 80% for proofs of all shown lengths. This is unsurprising, as these models would not be able to attain overall accuracies in the 90s without exhibiting stable performance across proofs of all sizes. Fig. 8 plots similar curves for the bottom 14 models. Here we see some steeper declines on longer proofs, but the overall effect remains relatively modest even in this group.

These plots suggest that the core challenge in checking these proofs is not their size. This aligns with our results from the harder task **PL3-PC**, where proofs are considerably longer yet performance remains largely insensitive to proof length, especially for top-tier models. And it is consistent with what we know about the computational complexity of proof checking, which is much lower—essentially $O(n)$ for a proof of size n —than that of reasoning problems involving search. One might therefore expect that these verification tasks could be scaled to longer proofs (with several hundreds or thousands of lines) without immediately plateauing the best models at a failure point, which could be useful for probing logical dependency management when the relevant information is spread out across far more tokens. However, the challenge version of this task that we introduce in Section 3.2, **PL1-PC-c**, will show that even the strongest models at present struggle in this task when the input proofs become significantly longer (in the hundreds of lines) and more structurally complex (e.g., with more heavily nested assumptions).

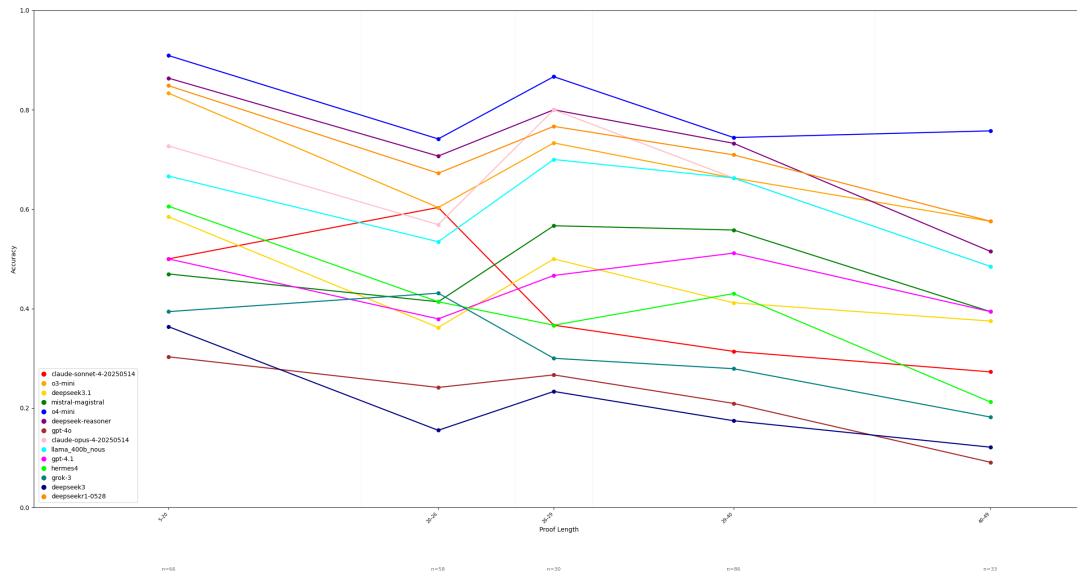


Figure 8. PL1-PC accuracies as a function of proof length for the bottom 14 models.

Qualitative Observations

PL1-PC outputs are richly structured and provide a wealth of data for analyzing performance and gaining deeper insights into how models reason. For instance, it is an interesting question whether specific types of model errors exhibit systematic correlations with structural proof characteristics such as the level of assumption nesting or recall span (how far back a cited lemma was derived). While we defer such finer-grained analyses to future work, in the rest of this section we make some high-level qualitative observations informed by a close inspection of the PL1-PC results.

- In this task, models that claim to find reasoning errors must justify their verdicts with detailed information about the alleged error—its exact location and characteristics. As mentioned earlier, the required documentation includes an *errorDescription* field whose value must describe the error in natural language. When less able models make a type-2 error, wrongly rejecting a proof as incorrect, their error descriptions not only tend to be vague, but they also tend to ramble on, often devolving into self-refuting statements. Instead of being concise error summaries, they end up serving as records of extended deliberations and self-analyses in which models eventually talk themselves out of their own error reports. This is a distinct linguistic signature of a recurring failure of cognitive control that manifests itself under different guises. For instance, recall the proof-writing pattern mentioned on p. 41 about comments being used as proof-search scratchpads, where the model contemplates different tactics and changes course midstream instead of commenting on the particular proof it has written, “failing to separate its internal monologue and search process from the final polished output.” Here is an example from GPT-4.1 on PL1-PC item 7, where the model claims to find an error in a proof that is in fact correct, objecting to this particular line:

```
B_or_A := (B | A) BY right-either on B, A_from_case1;
```

Here, *A_from_case1* is the atom *A*, which was previously derived and holds at this particular point in the proof. Recall also that *right-either* is a binary rule that takes any two formulas *p* and *q* as arguments, and provided that the second argument *q* holds, it returns the disjunction (*p* | *q*). GPT-4.1 explains its rationale for why the above application of *right-either* is flawed as follows (we reprint its response here verbatim, including the use of caps):

right-either requires its right argument to be a formula that currently holds, but at this point in the proof, B does not hold. The only formulas being assumed or derived inside this scope are (A & (A | B)), A (from left-and on case1), and A_from_case1 itself. B has not been established as true in this scope. The correct application

would be right-either on any formula for the left, and the right ($A_{\text{from_case1}}$), since right-either produces $(B \mid A_{\text{from_case1}})$, but requires only $A_{\text{from_case1}}$ to hold. However, using B as the left argument is only permitted syntactically, not semantically, if B does not hold; but the right argument ($A_{\text{from_case1}}$) must hold, which it does. However, right-either on B , $A_{\text{from_case1}}$ produces $(B \mid A_{\text{from_case1}})$, so as written, it is correct; but if the order were reversed, and the right argument did not hold, it would be an error. Thus, looking further, while the **BY** annotation and the use here is correct, it does not actually lead to a logic error at this specific step. Therefore, continuing to the surrounding context, the derivation in the second disjunct $((B \mid A) \& (B \implies \sim B))$ proceeds using $B_{\text{implies_notB}}$, which does hold. The next possible error is the application of `from-false` on $((\sim B \mid A) \& (\sim (B \mid (\sim A)))) \mid ((B \mid A) \& (B \implies (\sim B)))$ in the scope where false was just derived. However, `from-false` requires that false holds, which it does, and it can derive any formula, so this is also correct. Moving back to the **BY** annotation in the step:

```
conclusion_from_case1 :=
  (((~ B) | A) & (~ (B | (~ A)))) | ((B | A) & (B ==> (~ B)))
  BY right-either on (((~ B) | A) & (~ (B | (~ A))), D2
```

here, right-either requires only its right argument, $D2$, to hold, not the left; $D2$ is $((B \mid A) \& (B \implies \sim B))$, just constructed above. That is correct.

After fully checking, THERE IS NO LOGIC, TYPE, or SYNTAX ERROR in the annotated proof.

Nevertheless, the FINAL DERIVED CONCLUSION of the proof is the result of the last step:

```
cases on premise-1, case1_proof, case2_proof.
```

That produces the conclusion

```
((((~ B) | A) & (~ (B | (~ A)))) | ((B | A) & (B ==> (~ B)))).
```

which matches exactly the required conclusion! Thus, the proof is correct. Therefore, `errorDetails` should be empty and `correct` should be true.

Of course, it is incoherent that “`errorDetails` should be empty” when the model itself inserted this text into `errorDetails`.

This happens even with more recent and powerful models, though much less frequently. On problem 150, Grok-4 emits a page-long `errorDescription` that contains 10 occurrences of “wait”, 32 “yes” answers to questions that it posed to itself, 9 occurrences of “fine”, and ends with “Yes. Fine. I think the proof is correct.” Here is another one from Grok-4 on item 276:

The step derives $(\sim A)$ inside the assumption of B , but the use of `from-false` is valid only if false holds at that point. However, the previous contradiction is derived, yes, but the intention is to derive $\sim A$, which is fine, but wait, no error here. Wait, actually, upon reexamination, the error is later.

The meandering of these descriptions means that they’re often longer than their correct counterparts. For example, the mean string length of correct error descriptions by DeepSeek-V3 is 281, but when the model makes type-2 errors this jumps to 424, a 51% increase. For some models this increase is statistically significant⁴⁸ and could be used as a cheap signal to gauge the correctness of their outputs, essentially by treating the length of error descriptions as a lightweight proxy for uncertainty or reasoning instability, since longer outputs tend to reflect internal conflict in the reasoning trace, with the model alternating between competing internal judgments.

The fact that this is a cross-task behavioral motif suggests a broader hypothesis: LLMs exhibit cognitive leakage between what philosophers call the context of discovery and the context of justification (see Section 5 of Schickore’s article on scientific discovery (Schickore 2025)). When models are uncertain, explanations that are supposed to justify the reasoning behind their deci-

⁴⁸ We performed statistical comparisons between the mean character lengths of `errorDescription` strings for correct judgments and for type-2 errors (false rejections). For each model, we applied a one-sided permutation test (H_1 : type-2 > correct) together with Welch’s unequal-variance t -test and two effect-size estimates—Cliff’s δ and Hedges’ g . A difference was considered reliable when the permutation $p \leq 0.05$ and the 95% confidence interval of δ lay strictly above zero. By these criteria, several models show statistically significant increases in description length: DeepSeek-V3 (+143 characters, $\delta = 0.41$), claude-sonnet-420250514 (+86 characters, $\delta = 0.38$), grok-3 (+73 characters, $\delta = 0.29$), deepseek-r1-0528 (+71 characters, $\delta = 0.52$), and o3-mini (+102 characters, $\delta = 0.54$). For stronger models with very few type-2 cases (e.g., grok-4 and claude-sonnet-4.5-ext-thinking), the apparently large jumps are statistically unreliable due to small-sample variance.

sions sometimes collapse into stream-of-consciousness attempts to second-guess and revise those decisions, typically ending in a state of cognitive dissonance.

- Some error explanations are incoherent not because the model contradicts itself on the existence of the error, but for other reasons, such as gross pattern-matching mistakes or random unintelligible claims. Here is an example from deepseek-reasoner (on item 163):

dn must be applied to a double negation of the form $(\sim \sim p)$, but here it is applied to $(\sim (\sim (\sim C \ \& \ A)))$, which is a triple negation and does not match the required syntactic form.

Brushing aside the fact that every triple negation is also trivially a double negation, the formula in question here, $(\sim (\sim (\sim C \ \& \ A)))$, is not a triple negation; the innermost negation sign applies only to C, unlike the two previous negation signs.

- All models, including the best ones available today, seem to have a hard time grasping rudimentary concepts involving negation. A notable example is a widespread failure to understand complementation. This is a pervasive notion in logic, especially in the context of SAT solving, where it is usually defined for CNF literals: the complement of an atom α is $\neg\alpha$, while the complement of a negated atom $\neg\alpha$ is α . Thus, unlike conventional syntactic negation, which always prepends a negation sign, complementation is an involutive operation that systematically flips polarity. If we write $c(l)$ for the complement of a literal l , we always have $c(c(l)) = l$. The notion is naturally extended to all formulas: if a formula p is a negation $\neg q$, then its complement is q ; and if p is not a negation, its complement is $\neg p$. The concept is introduced in the first paragraph of the **PL1-PC** input prompt as a relation, emphasizing that it is a symmetric relation:

We say that two formulas are “complementary” (or “complements”, or that they “complement” each other) when one is the syntactic negation of the other. This is a symmetric relation: p complements q iff q complements p .

Accordingly, A complements $\neg A$ just as well as $\neg A$ complements A . There is no directionality or order involved; it's not the first argument that always needs to be the negation of the second, or the other way around. The only requirement is that one of the two (either one) is the negation of the other.

Certain inference rules of NDL require some of their arguments to be complementary. For instance, **mt** (modus tollens) must be applied to two formulas of the form $(p \Rightarrow q)$ and r where r and q are complements. Assuming that both $(p \Rightarrow q)$ and r hold, the result (conclusion) is $\neg p$. Likewise, **dsyl** (disjunctive syllogism) is applied to two formulas of the form $(p_1 \vee p_2)$ and q , both of which must hold and where q complements one of the disjuncts, either p_1 or p_2 . If it complements p_1 then the result is p_2 , and if it complements p_2 the result is p_1 . This is not a subtle notion, yet somehow all models end up getting it wrong at some point (some more often than others), frequently confusing it with negation, which is intrinsically asymmetric. Here is an example from Grok-4 on item 56:

dsyl requires the second argument to complement one of the disjuncts in the first argument syntactically, but D does not complement $(\sim D)$ or $(C \Rightarrow A)$. The syntactic complement of $(\sim D)$ is $(\sim (\sim D))$, which is not identical to D.

But if “the syntactic complement of $(\sim D)$ is $(\sim (\sim D))$,” then the syntactic complement of D must be $(\sim D)$. And if complementation is symmetric, as stated in the prompt, then D also complements $(\sim D)$. The rudimentary error here is to view complementation as identical to negation.

As another example, here is part of the error description that GPT-5 gives for item 184:

*dsyl requires arguments of the form $(p \mid q)$ and r where r is complementary to either p or q . Here **disj1** is $(A \mid \sim B)$ and the second argument is B, which is not complementary to either A or $\sim B$. Therefore the application of dsyl is invalid.*

Similar mistakes are common, e.g., GPT-5.2 says the following regarding item 36:

*This application of dsyl is invalid. The disjunction **notA_or_notnotB** is $((\sim A) \mid (\sim (\sim B)))$. Disjunctive syllogism requires the second argument to complement (i.e., be the syntactic negation of) one of the disjuncts. But **not_B** is $(\sim B)$, which does not complement either $(\sim A)$ or $(\sim (\sim B))$ (the complement of $(\sim (\sim B))$ would be $(\sim (\sim (\sim B)))$). Therefore dsyl cannot be applied here, and the conclusion $(\sim A)$ is not derivable by this step.*

Likewise, on problem 69 it states that $(\sim B)$ does not complement $(\sim \sim B)$, and on problem 198 it stresses that $(\sim A)$ is not complementary to $(\sim \sim A)$.

- The most noteworthy and concerning theme that emerges from a close inspection of the data is a recurring failure of models to exhibit judgment consistency. A model might flag something as an error in one proof and disregard the exact same issue in another proof. We refer to this broad phenomenon as *epistemic instability*. It has many manifestations, with one of the most striking instances emerging from the interaction between proof writing and proof checking. For example, a model might write a correct proof D for a given problem, but when we later ask it independently to verify D , it incorrectly rejects it as wrong. This is clearly problematic, suggesting a sort of split-brain syndrome whereby the model writes proofs with one brain and checks them with another.

For a human reasoner, it would be odd if a person who is both rational and truthful would write a correct proof and then reject that same proof, especially if this behavioral cycle repeated itself. After writing their proof for the first time, they might perhaps discover what they (incorrectly) believe to be a bug. But if so, they would not then go on to rewrite the same proof with the same alleged bug. In other words, even if a human reasoner exhibited an initial inconsistency between the two tasks, they would quickly reach an equilibrium. Crucially, this equilibrium is independent of the proof's objective correctness; it is instead determined by the human reasoner's subjective cognitive state. Unless they were trying to be deceptive, their proof-writing and proof-checking stances would align. Indeed, it would be a sign of incoherence if they did not. But we found such unstable cross-task behavior to be a common motif with AI models.

We plan to revisit model behavior at the interplay between proof writing and proof checking in forthcoming work. However, the phenomenon is not limited to the interaction between those two tasks. A careful analysis shows that it is frequently seen even within **PL1-PC** in isolation. An example is given by the aforementioned issue with complementation. For every time that a model flags a spurious error by demanding syntactic negation instead of complementation, there are more times where it does *not* flag instances of the exact same pattern, seemingly happy to accept complementation instead of strict syntactic negation. This indicates that the model has no stable epistemic commitment on the issue and no coherent internal principles governing the distinction between the two.

For concrete examples, recall that on multiple occasions GPT-5.2 insists on conflating complementation with syntactic negation. Yet many other times it does the opposite. For instance, on item 24, it condones an application of `dsy1` to $(\neg A \vee \neg C)$ and C , even though, according to its own logic, "disjunctive syllogism requires the second argument to complement (i.e., be the syntactic negation of) one of the disjuncts" and C is obviously not complementary to $\neg C$ in that sense. Likewise, on item 37, GPT-5.2 happily accepts an application of `dsy1` to $(\neg \neg C \vee \neg B)$ and B . On item 49 it accepts an application of `dsy1` to $((\neg A \Rightarrow C) \vee \neg(D \wedge E))$ and $(D \wedge E)$. On item 15 it accepts an application of `dsy1` to $((A \Rightarrow \neg B) \vee \neg(C \Rightarrow B))$ and $(C \Rightarrow B)$. There are other examples. On item 299 it delivers a lecture on how B does complement $\neg B$, even though B is clearly not the syntactic negation of $\neg B$.

As another example, GPT-5 rightly complains several times about applications of `dm` (De Morgan) that skip a double negation step. For instance, on item 171 it writes:

*Applying dm to $(\sim (\sim A \& C))$ should yield $(\sim \sim A \mid \sim C)$, not $(A \mid \sim C)$. The step's **BY** annotation claims a conclusion that is not the direct result of the dm rule (it eliminates the double negation without a separate dn step).*

GPT-5 flags the same error elsewhere (e.g., items 143 and 210). Yet on item 298 it happily accepts the following fragment:

```
...
assume h3 := ( $\sim (\sim B \& A)$ ) {
  ( $B \mid \sim A$ ) BY dm on h3;
...

```

```
}
...
```

even though this dm application is structurally identical to (and makes the exact same error as) the one in item 171.

Here is another example: As we pointed out earlier (p. 4), LLM-written proofs often write conditional proofs with syntactically malformed assumptions, along the lines of `assume arbitrary-identifier { ... }`. In **PL1-PC**, strong models often detect this error and explain it quite well. For instance, here is what GPT-5 says about the proof of item 140:

Malformed assume statement. A conditional proof must have the form

$$\text{assume } p \{ \dots \}$$

or

$$\text{assume } I := p \{ \dots \}$$

where p is a formula. Here, `assume B_hyp` lacks a hypothesis formula (and `B_hyp` is not defined as a formula), so the step cannot be parsed and the intended conditional ($B \implies C$) is not formed.

But on item 281 GPT-5 says nothing about the conditional subproof

```
assume B_hyp { C BY claim on C; };
```

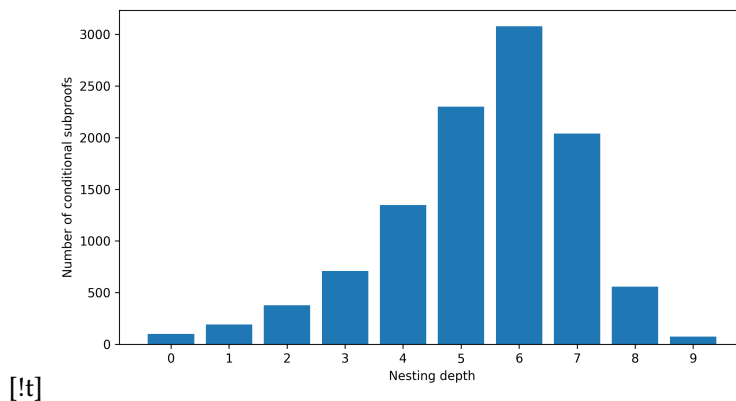
even though the assumption is malformed in exactly the same way as the one that GPT-5 complained about in item 140. The same pattern can be observed across all systems; we use examples from GPT-5 and GPT-5.2 here for illustration purposes.

These inconsistencies are akin to a teacher grading student assignments who sometimes marks a particular step as an error (“you can’t use double negation here”) and other times gives full credit for the very same step. The problem isn’t the strictness (or even the fairness) of the grading, but the *judgment instability*: the teacher’s evaluative criteria drift from one paper to the next, suggesting they have no stable conception of what the rules actually mean.

It is possible that these issues will be resolved in the near future, although the pervasive judgment inconsistencies point to deeper problems of epistemic coherence; we will return to this topic in Section 7.5. Our main aim in this section is not to catalog a particular set of failure modes or quirks of these specific model versions, which are likely to be transient, but to illustrate the sort of reasoning pathologies that tasks like **PL1-PC** can expose. These clearly demonstrate the need for richer, non-scalar evaluation signals for assessing reasoning competence.

A Challenge Version of PL1-PC

In this section we present results for **PL1-PC-c**, a more challenging proof-checking dataset consisting of 50 longer proofs. We only computed results for the 4 currently strongest models, as the rest are very unlikely to perform better than random guessing. Even among these 4 top-tier models, only two of them manage to perform better than flipping a coin on the guess-only version of the task. These 50 proofs have lengths ranging from 720 to 987, with average and median lengths of 839 and 823, respectively. The proofs were generated by our own hand-rolled theorem prover and were verifiably correct. We then randomly and minimally corrupted a small number of them (by injecting exactly one error in 9 of the 50 proofs). The proofs are challenging to verify because they have a lot of conditional subproofs (215.5 per proof on average), with a high degree of nesting. The distribution of nesting depths is shown below.



The figure shows that the mass of conditional subproofs rises steadily and peaks around depth 6, then falls off fairly quickly after 7. So a typical subproof in this challenge corpus is nested fairly deeply (at an average depth of 5.3).

Results for the 4 models are shown below.

Model	Full Task	Guessing Only	Model	Full Task	Guessing Only
claude-opus-4-5-20251101	0.65 (± 0.13)	0.69 (± 0.12)	gpt-5.2	0.64 (± 0.13)	0.66 (± 0.13)
gemini-3.1-pro-preview	0.48 (± 0.13)	0.52 (± 0.13)	gpt-5.4	0.44 (± 0.13)	0.48 (± 0.13)

Table 6. PL1-PC-c accuracies, for the full task and for simple guessing.

All models perform somewhat better in the guessing-only version of the task, but the deltas are marginal. This implies that the main difficulty for these models is deciding correctness in the first place, not localizing and explaining errors. Restricting attention to the guessing-only task variant, even the top-performing model, Claude Opus 4.5, performs only 38% better (69-50/50) than random guessing.⁴⁹ It does worse compared to the majority-guess baseline: because 41 out of the 50 proofs are correct, always guessing “correct” guarantees an accuracy of 82%, considerably higher than 69%, which suggests that the model is not exploiting the class imbalance. Indeed, all four models are below the majority baseline. If they were even weakly exploiting signal in the proofs, they should at least conclude that most proofs are correct. The fact that they don’t suggests that they are not reliably extracting the right structural cues. Remarkably, both Gemini 3.1 and GPT-5.4 fail to beat random guessing.

Perhaps the clearest takeaway from these results is that proof-checking/debugging ability is an orthogonal capability axis. While Claude Opus 4.5 places last among these 4 elite models in generative reasoning tasks like proof unmasking and gap filling (see Sections 4 and 5), it takes first place in proof checking. Likewise, even though GPT-5.4 outperforms GPT-5.2 in those tasks, here it trails behind by 20 absolute points, reinforcing the point that better generation does not translate to better verification.

4. Proof Masking

In this section we introduce a novel proof-completion task. Given a proof, we mask any of the following four types of items:

1. conclusions (formulas immediately preceding the **BY** keyword);
2. inference rules (immediately preceding **on**);
3. assumptions (immediately following **assume**); and
4. one or more arguments to an inference rule application (immediately following **on**).

⁴⁹ Flipping a coin still guarantees 50% accuracy, even though the set is imbalanced with 41 correct proofs and 9 incorrect ones.

Masks are unique identifiers: MASK1, MASK2, and so on (though the indices need not be consecutive). We only mask formulas at the top level, not any proper subformulas (e.g., we are never left with something like $(\sim \text{MASK8} \mid C)$).

The objective is to determine if there is a way to unmask the proof so that it correctly derives the target conclusion from the given premises; and if so, to provide a detailed assignment of appropriate values to the masks. As an example, consider again this proof:

```

1 assert premise-1 := (A ==> B)
2 assert premise-2 := (~ A ==> C)
3 assert premise-3 := (C ==> D)
4
5 # We prove (B | D) by a case analysis on (A | ~ A), which holds by the law of the excluded middle.
6 {
7   (A | ~ A) BY ex-middle on A;
8
9   # Case 1: Show that A implies (B | D)
10  assume A {
11    # Applying modus ponens to premise-1 and the assumption A gives B:
12    B BY mp on premise-1, A;
13    # So now (B | D) follows by disjunction introduction:
14    (B | D) by left-either on B, D
15  };
16
17  # Case 2: Show that (~ A) also implies (B | D)
18  assume (~ A) {
19    C BY mp on premise-2, (~ A);
20    D BY mp on premise-3, C;
21    (B | D) BY right-either on B, D
22  };
23
24  # The case analysis is now complete:
25  (B | D) BY cases on (A | ~ A), (A ==> B | D), (~A ==> B | D)
26 }

```

Here is a masked version of this proof (after removing comments):

```

1 assert premise-1 := (A ==> B)
2 assert premise-2 := (~ A ==> C)
3 assert premise-3 := (C ==> D)
4 # Goal: (B | D)
5
6 {
7   (A | ~ A) BY ex-middle on MASK1;
8   assume A {
9     MASK2 BY MASK3 on premise-1, MASK4;
10    (MASK5 | D) by left-either on MASK6, MASK7;
11  };
12  assume MASK8 {
13    MASK9 BY mp on premise-2, MASK10;
14    D BY mp on MASK11, MASK12;
15    MASK13 BY right-either on B, D;
16  };
17  MASK14 BY cases on (A | ~ A), MASK15, MASK16
18 }

```

With the original proof in sight, it is of course straightforward to generate a correct mask assignment: MASK1: A; MASK2: B; MASK3: mp; MASK4: A; and so on. But without the benefit of having seen the proof before, this is not trivial. It can be loosely likened to a constraint satisfaction problem (CSP), where the variables are the masks and the constraints are given by the syntax rules as well as the operational semantics of the proof language, including scope rules (for names as well as assumptions and intermediate conclusions). For instance:

- We can infer that MASK1 must be A from the specification of the excluded-middle rule and the fact that we see the conclusion $(A \mid \sim A)$ to the left of **BY**.
- On the last line, because the goal is $(B \mid D)$ and the pivot disjunction is $(A \mid \sim A)$, we can infer from the specification of cases that MASK15 and MASK16 must be conditionals of the form $(A ==> (B \mid D))$ and $(\sim A ==> (B \mid D))$, respectively.

- Knowing that the high-level proof is a case analysis on $(A \mid \sim A)$, we can infer that the conditional subproof on lines 12–16 must postulate $(\sim A)$ as an assumption, i.e., that MASK8 must be $(\sim A)$.

When a proof is heavily obscured, the problem generally becomes more complex. In this dataset, we randomly masked anywhere from 30% to 90% of all candidates (of all 4 types listed above) in a given proof.

Although this task is superficially similar to a CSP, it diverges in two crucial respects. In a CSP, constraints are explicitly given as part of the problem: each constraint pertains to a specific list of variables and allows a specific set of assignments (tuples of values). By contrast, the constraints governing valid mask assignments are implicit, heterogeneous, and distributed across the proof's syntactic and semantic structure. Indeed, identifying these constraints is a key part of the problem. The main source of difficulty is not combinatorial solution multiplicity but delayed constraint revelation and the need to maintain a long sequence of correct commitments. The task thus occupies a regime where symbolic reasoning is unavoidable but the relevant symbolic structure is not explicitly given. This distinguishes it from puzzles where the main difficulty lies in combinatorial search rather than in extracting the relevant constraints from context and the operational semantics of proofs. When models fail to unmask a proof, it is not simply because the search space is large, but because the task requires jointly inferring structure, constraints, and solutions under global consistency requirements. This reflects deeper limitations than simply “models can't do backtracking.” The second major difference is that the set of values is not explicitly given either. It is not even clear that this set is finite. We know that mask values are either formulas or rules, and that the former are likely to be subformulas of the premises and/or the goal, of which there is only a finite number. But the subformula property is not guaranteed, since we know nothing about the original proof, which could have arbitrary detours above and beyond the usual and relatively innocuous detours introduced by classical logic (double negations and the use of false). A fortunate corollary of these differences is that **PL1-PM** instances cannot be straightforwardly encoded as CSP or SAT problems and outsourced to an external solver. Nor is it easy for a model to solve the task just by writing bespoke code for it. Even if the model could write flawless infrastructure for the problem (e.g., a parser for masked NDL proofs), an unmasking algorithm would still be far from obvious.

However, even with a large number of masks sprinkled over a large proof, this is a relatively easy task as long as mask *density* is not excessive (see below for a more precise definition of density). First, because NDL is strongly structured, the expected category (or “type”) of a mask is fixed: it is either *formula* or *rule*, and it is determined by the position of the mask variable: if the mask occurs immediately before a **BY** or immediately after an **assume**, it must be a formula; if it occurs immediately before an **on**, it must be a *rule*; and so on. Second, all masks are unique, so there are no cross-references and unification across multiple proof lines is not required. Finally, most masks are functionally determined and have unique/canonical values that can be locally inferred. Accordingly, putting aside pathological cases such as proof with several steps that are completely masked, there is no combinatorial explosion of solutions, no need to keep track of large equivalence classes of assignments, and no true symmetry-breaking issues (aside from rare commutations like example 3 below). This means that values for most masks can be inferred incrementally, on a line-by-line basis. Some choices are, of course, semantically coupled across the entire proof tree. For example, choosing the conclusion of an excluded middle application constrains the shape of the subsequent case analysis and thus the assumptions and conclusions of the relevant conditional subproofs and the arguments of the corresponding cases application. But these tend to be light inter-dependencies that become obvious after a single pass over the proof tree, which typically suffices because the constraints are acyclic (determined by the linear order of the proof)⁵⁰ and nearly deterministic.

⁵⁰ A proof step depends only on information established in earlier steps (such as previously derived conclusions, introduced premises, or assumptions) and never on content that appears later. This means that the constraint graph over masks is a DAG (directed acyclic graph), allowing the proof to be reconstructed or verified in a single forward pass without backtracking. Once a line has been correctly unmasked, it can be referenced but need not be altered. In that sense, the task is topologically sorted by proof order: solving or checking constraints proceeds deterministically from top to bottom.

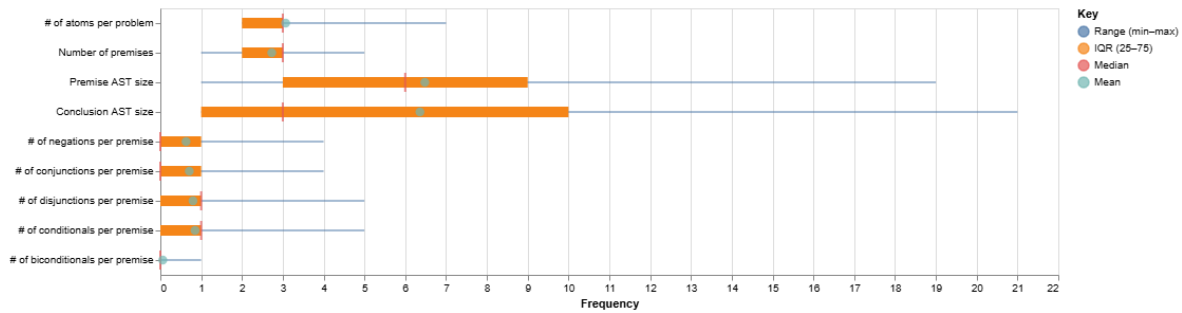


Figure 9. PL1-PM problem statistics

The **PL1-PM** task slice contains 300 proofs that have been masked as described above. All 300 proofs are correct, containing no errors of any kind. 80 of these proofs are conditionalized, meaning that the relevant premises are explicitly assumed as hypotheses (instead of taken as global premises). Statistics about the corresponding problems are shown in Fig. 9. Note that these are “easy” problems along pretty much all dimensions: the total number of premises is small (less than 3 on average) and, viewed as an AST, each premise has about 6 nodes on average. Conclusions have a median size of 3 nodes but the average is considerably higher (slightly over 6). The median number of propositional atoms in a problem is 3, and negations are the most rare logical connective save for biconditionals.

Statistics about the length of these proofs are shown in the following table. Note that these proofs are even shorter than those in **PL1-PC**, whose median length is 16% greater. 90% of the **PL1-PM** proofs have fewer than 50 lines.

	Mean	Median	Min	Max	Std	@5	@25	@50	@75	@90	@95
Proof length	26.81	25.0	3	64	14.12	6.0	15.0	25.0	38.0	47.0	50.0

More granular statistics about the distribution of inference rules and conditional reasoning in these proofs can be found in Fig. 10. Conditional (hypothetical) reasoning is very common in this corpus, largely because about half of these proofs are conditionalized and all such proofs start by explicitly making hypotheses.

Given a masked proof and the underlying argument (premises and target conclusion), the model is asked to output a dictionary M that maps every mask in the input to an appropriate value:

$$\{\text{MASK1} : \dots, \dots\}$$

in such a way that when the proof is unmasked in accordance with M (i.e., by plugging in $M[m]$ for every mask m that appears in the input), the result is a complete proof that correctly derives the goal from the premises. The model is told to output the empty dictionary $M = \{\}$ if it believes that there is no valid assignment of values to masks that can successfully unmask the proof. This effectively acts as a distractor, since we know that every masked proof can be unmasked.

The key question addressed by this task is whether a model can infer missing symbolic structure consistent with the rules that govern natural deduction. Whereas proof writing and proof checking test a model’s ability to construct or evaluate a line of reasoning, proof masking asks it to reconstruct elided reasoning details from partial information, inferring what intermediate conclusions, assumptions, or rule applications could make the surrounding proof context coherent. This task is superficially reminiscent of code infilling, where a model must recover a missing span of source code, but the resemblance is limited. In code infilling the missing span typically forms a single contiguous “hole”—a bulk fragment whose boundaries are arbitrary and syntactically defined, and whose recovery in practice depends mainly on local lexical and type constraints. By contrast, in proof masking the missing content is dispersed across dozens of micro-holes, each corresponding to a distinct atomic element of the proof (an individual conclusion, rule name, assumption, or argument to a rule application). Each mask encodes only a tiny unit of inferential information, yet all of these units must be filled

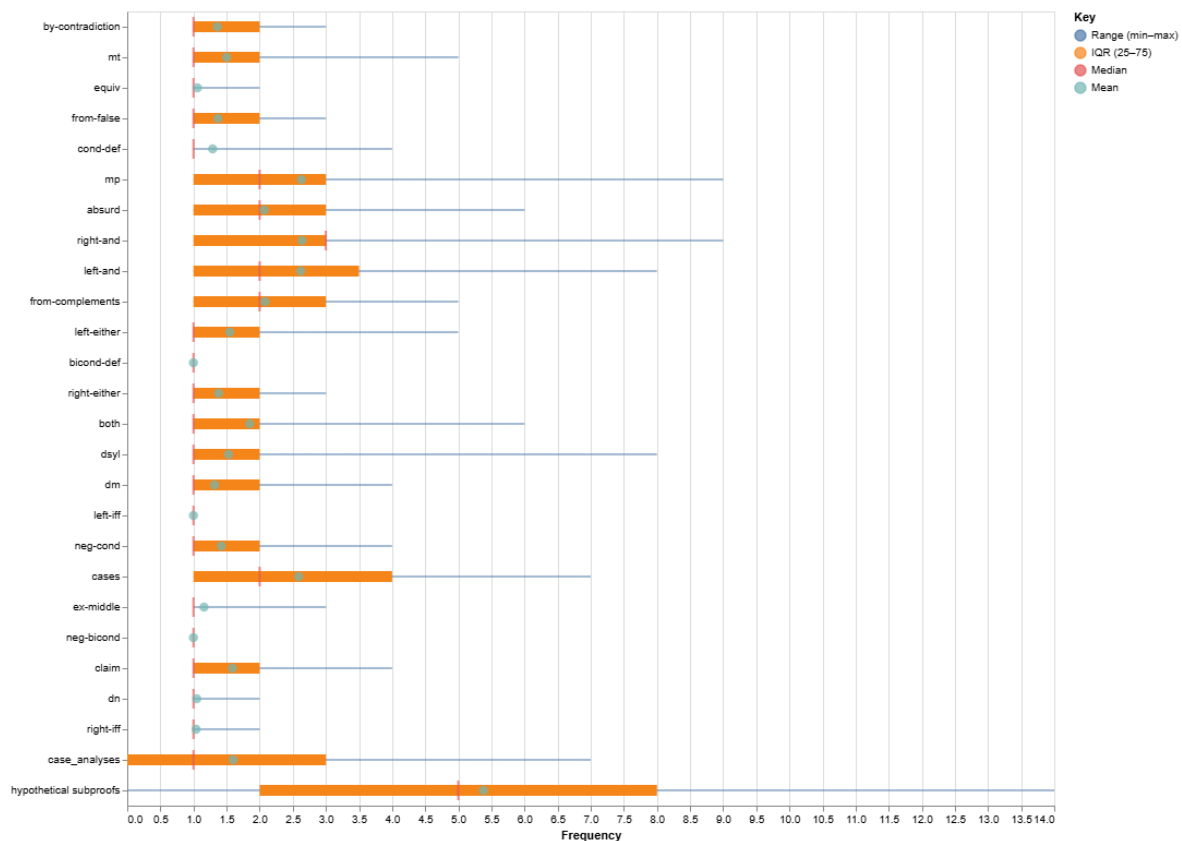


Figure 10. Inferential profile of PL1-PM proofs.

jointly and coherently, because the value assigned to one mask might constrain the permissible values of subsequent masks. The model must therefore maintain a global, internally consistent hypothesis about the entire proof structure while simultaneously resolving many fine-grained local uncertainties. Essentially, it must treat the proof itself as an object of reasoning, reasoning about reasoning rather than merely executing it. This gives proof masking a distinct meta-reasoning flavor. A system that can perform well in this task demonstrates not only procedural knowledge of inferential mechanisms, but also a representational grasp of how those mechanisms fit together and a capacity for metalogical comprehension.

As we already pointed out, most mask values are uniquely determined. In general, however, masking problems are underdetermined, i.e., there may be multiple mask assignments that yield a correct proof. Accordingly, it is not expected that a mask assignment produced by a model recovers the original proof. For instance, the original proof might contain a chain of two consecutive applications of left and right conjunction elimination:

$$\dots \text{left-and on } (A \& B); \text{right-and on } (A \& B); \dots \quad (3)$$

But if we mask both rules, e.g., by replacing `left-and` by `MASK-3` and `right-and` by `MASK-4`, then `MASK3: left-and`; `MASK4: right-and` and `MASK3: right-and`; `MASK4: left-and` are both valid solutions, because the two inference steps can be swapped without affecting the correctness of the subsequent reasoning. For that reason, LLM responses must be evaluated semantically rather than syntactically. That is, LLM responses are evaluated not by matching the specified mask values against a fixed ground truth, but by unmasking the proof according to those values and then using Athena to check that the resulting proof correctly derives the target. If a model claims that the problem is unsolvable, that response is always counted as incorrect, since we know by construction that every masked proof can be unmasked.

It may be interesting to note that while the reasoning required for this task appears to be purely deductive, it actually leans heavily on pragmatics and on inductively learned regularities about proof

construction. In particular, many proof steps are not strictly (deductively) determined by the local context, but they *are* determined when one takes into account how proofs tend to be written in practice. As a concrete example (from item 5 of the challenge dataset), suppose the proof has a biconditional premise of the form $(p_1 \Leftrightarrow p_2)$, say $((D \Rightarrow B) \Leftrightarrow (B \Leftrightarrow A))$. The first three steps of the masked proof are as follows:

```

1 MASK2 BY MASK1 on ((D ==> B) <==> (B <==> A));
2 p0 := ((B <==> A) ==> (D ==> B)) BY MASK3 on ((D ==> B) <==> (B <==> A));
3 ((~ (D ==> B)) | (B <==> A)) BY MASK5 on MASK4;

```

It is clear from line 2 that MASK3 must be *right-iff*, as that is the only inference rule that could produce the given conclusion. Now, in theory, line 1 remains underconstrained, in that MASK1 could still be either *left-iff* or *right-iff*. This is because, in principle, the proof could contain a redundancy and go on to repeat the same inference on line 2; nothing in the logic forbids that. *Pragmatically*, however, this would be highly unorthodox. It's much more likely that line 1 detaches the left-to-right part of the biconditional premise and line 2 detaches the right-to-left part. That's not a quirk; it's an inductive regularity about proofs as social artifacts, not just as logical objects. So a human solver would naturally commit to MASK1 being *left-iff* after deducing from line 2 that MASK3 must be *right-iff*.⁵¹ This marks another point of divergence from CSP/SAT problems, which do not have any notion of pragmatics. The upshot is that proof reconstruction in this task, at least for humans, relies not just on pure deduction but on proof-writing conventions, stylistic regularities, and general inductively derived pragmatic expectations about proof structure.

Model	Instrumented Accuracy	Strict Accuracy	Model	Instrumented Accuracy	Strict Accuracy
gemini-3.1-pro-preview	0.98 (± 0.02)	0.97 (± 0.02)	gemini-3.0-pro-preview	0.96 (± 0.02)	0.93 (± 0.03)
gpt-5.2	0.90 (± 0.03)	0.82 (± 0.04)	claude-opus-4-5-20251101	0.89 (± 0.04)	0.87 (± 0.04)
gpt-5.1	0.89 (± 0.04)	0.84 (± 0.04)	grok-4	0.82 (± 0.04)	0.68 (± 0.05)
gemini-2.5-pro	0.81 (± 0.04)	0.50 (± 0.06)	gpt-5	0.79 (± 0.05)	0.64 (± 0.05)
claude-sonnet-4.5-ext-thinking	0.79 (± 0.05)	0.70 (± 0.05)	o3	0.75 (± 0.05)	0.69 (± 0.05)
deepseek-reasoner	0.60 (± 0.06)	0.53 (± 0.06)	deepseek-r1-0528	0.54 (± 0.06)	0.37 (± 0.05)
o4-mini	0.49 (± 0.06)	0.40 (± 0.06)	claude-opus-4-20250514	0.40 (± 0.06)	0.30 (± 0.05)
o3-mini	0.31 (± 0.05)	0.16 (± 0.04)	claude-sonnet-4-20250514	0.29 (± 0.05)	0.23 (± 0.05)
deepseek-v3.1	0.25 (± 0.05)	0.19 (± 0.04)	grok-3	0.19 (± 0.04)	0.13 (± 0.04)
deepseek-v3	0.17 (± 0.04)	0.12 (± 0.04)	hermes-4	0.15 (± 0.04)	0.10 (± 0.03)
llama-3.1-405b	0.13 (± 0.04)	0.07 (± 0.03)	gpt-4.1	0.09 (± 0.03)	0.06 (± 0.03)
mistral-magistral	0.06 (± 0.03)	0.05 (± 0.03)	gpt-4o	0.05 (± 0.03)	0.04 (± 0.02)

Table 7. PL1-PM accuracies, using both the instrumented NDL checker and the regular proof checker.

Results are shown in Table 7, both under the instrumented proof checker and the conventional one. Without the benefit of instrumentation, performance tends to drop (significantly in many cases), though the vast majority of these errors are uninteresting and rather innocuous, such as omitting an outer pair of parentheses from formulas.

On the basis of the instrumented accuracies of Table 7, the discriminability index (DI) of **PL1-PM** is 0.38. As discussed in Section 3.2, this means that, on average, any two models differ in accuracy by about 38 percentage points—a very large spread showing that the task teases apart models of different ability levels extraordinarily well.

While the leading 3 models attain impressively high scores on this task, it should be kept in mind that the proofs in this dataset are very short (the median length is 25 lines, while 95% of the proofs have no more than 50 lines). Even with these very short proofs, Fig. 11 shows that the performance for most of the top 10 models drops as proof length increases even modestly, although the drop is very

⁵¹ Deductively, there is yet another possibility for the first line: line 1 could be an application of “biconditional definition” that derives $((p_1 \wedge p_2) \vee (\neg p_1 \wedge \neg p_2))$ from $(p_1 \Leftrightarrow p_2)$. But this would also be dispreferred by human subjects, because we implicitly know, for example, that proofs typically use a connective before expanding its definition, and that canonical pairs of eliminations (e.g., a *left-iff* immediately followed by a *right-iff*) are an extremely common inference pattern that is stylistically tight.

gentle for the top 4 models (and the performance of Gemini 3.1, in particular, is essentially flat). As shown in Fig. 12, the decline is much more dramatic for the remaining 14 models, whose accuracies effectively drop to zero by the time we get to proofs with a mere 30-35 steps, with only a couple of exceptions.

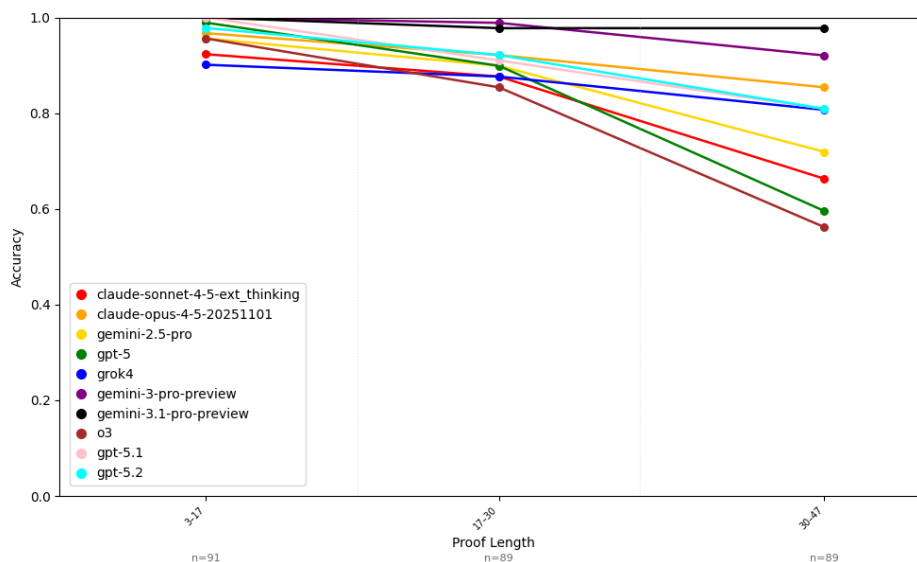


Figure 11. PL1-PM accuracy vs proof length for the top 10 models; variable-width bins with $n \geq 25$.

This performance profile should be expected when correctness requires all masked positions to be filled simultaneously.⁵² If each individual prediction has probability p of being correct, then the probability that the entire proof is correctly unmasked scales approximately as p^M , where M is the number of masks. Thus, even small per-mask error rates compound multiplicatively into rapid instance-level degradation. In addition, longer proofs impose heavier burdens on transformer attention and retrieval, with the model having to track more assumptions, rule dependencies, and line references,

⁵² The word “simultaneously” is referring to the evaluation criterion, not to the generation process. The models of course generate mask values autoregressively, but the correctness of their outputs is evaluated over the entire set of masks jointly, with an output counting as correct only if all masks are given valid values that collectively yield a correct proof. So even though generation is incremental, the reward landscape is globally conjunctive, with success requiring a perfect joint configuration of *all* micro-decisions made by a model.

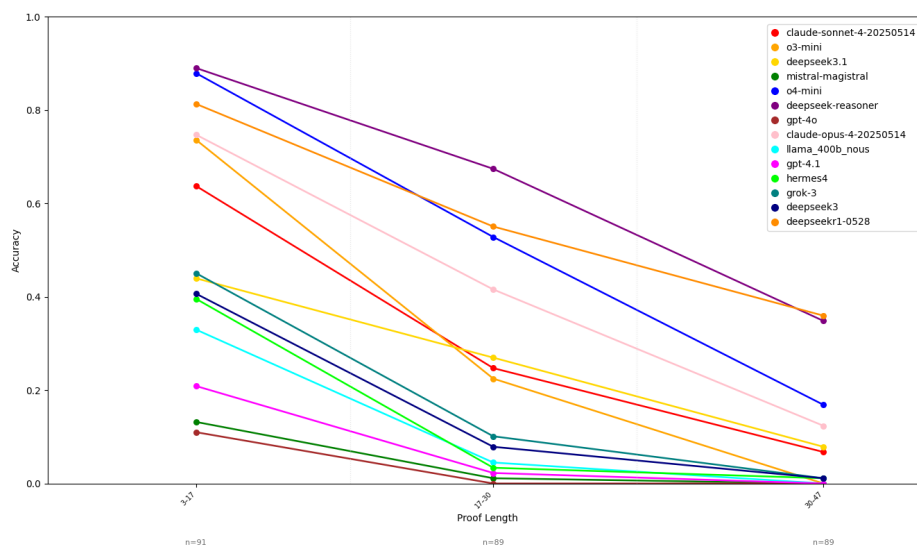


Figure 12. PL1-PM accuracy vs proof length for the bottom 14 models; variable-width bins with $n \geq 25$.

while maintaining perfect syntactic fidelity. These factors make the task more fragile under sequence generation.

A similar multiplicative-error-compounding argument could be made for proof checking, especially for correct proofs: on the assumption that a proof step has an independent fixed probability p of being correctly checked by a model, proof-checking accuracy should decay exponentially with longer proofs even if the task is locally easy (has a high p). However, what we saw in **PL1-PC** is that accuracy either held steady or declined more gracefully as proof length increased. So why does proof-checking seem to scale better than proof masking? While the two tasks share some similarities (both are “locally easy”), they also place different demands on a generative model. Given an incorrect proof in **PL1-PC**, a model must report the first erroneous step, which can be viewed as a sequential test over steps $1, \dots, N$ with success probability

$$P(\text{success} \mid j^*) = (1 - f)^{j^* - 1} (1 - m),$$

where we write j^* for the location (line number) of the first error, and where f is the false-positive rate on correct steps above j^* (i.e., the probability that the model will incorrectly mark such a correct step as erroneous) and m is the miss rate on that first error. For strong models, and assuming we are dealing with average incorrect proofs that have not been adversarially selected, f and m are small and approximately length-invariant, so the probability of success will be relatively high even for a long proof.⁵³ For *correct* proofs, on the other hand, the proof-checking situation is mathematically similar to that in proof masking, with both tasks exhibiting multiplicative structure. In particular, writing P_c and P_m for the **PL1-PC** and **PL1-PM** probabilities, respectively, we have $P_c(\text{success}) = p^N$, where $p = 1 - f$ is the probability that a step is correctly checked and N is the size of the proof (number of steps). Likewise, $P_m(\text{success}) = q^M$, where q is the probability of correctly decoding a single mask and M is the total number of masks in the proof. While both expressions have the same form, exponential decay is highly sensitive to base rates, so small differences in those rates can translate into big differences in performance. For instance, if $p = 0.999$ (i.e., the error rate is $f = 0.1\%$ per step) and $q = 0.99$ (with a 1% error rate per mask), then for $N = M = 60$ we have $P_c(\text{success}) = 0.94$, whereas $P_m(\text{success}) = 0.54$. Why might the proof-masking base rate be an order of magnitude smaller than the proof-checking base rate (for correct proofs)? Because one task requires generation while the other only requires recognition: in **PL1-PM**, a model must produce syntactically correct formulas, variables, and rule names, whereas for **PL1-PC** it only needs to judge correctness, without any risk of generation errors.⁵⁴

Error Analysis

There is no natural error typology for **PL1-PM** that partitions all possible errors into a set of mutually exclusive and jointly exhaustive types that can be readily counted, so we instead compute statistics on a number of revealing error types that are analytically meaningful but neither exclusive nor exhaustive. These classes of errors are described below. It will be useful to define μ as a shorthand for the following quantity: Writing $\#(D)$ for the number of masks that occur in a masked proof D , and if D_1, \dots, D_{300} are the proofs in **PL1-PM**, we set:

$$\mu = \sum_{i=1}^{300} \#(D_i).$$

1. *illFormattedAnswers*: These occur when the output generated by a model is not a JSON dictionary of the schema specified by the prompt (a mapping from masks to appropriate values). Min value: 0, max value: 300.
2. *emptyAnswers*: These errors occur when a model returns the empty dictionary $\{\}$ as its answer, indicating that there is no way to unmask the input proof. (Recall that the possibility of an empty

⁵³ For instance, with $f = 0.001$ and $m = 0.01$ we get $P(\text{success} \mid j^*) = 0.9$ even for $j^* = 100$.

⁵⁴ This applies to classification-only formulations of proof checking, where the model does not need to generate any structured output.

answer was included as a distractor, since we know by construction that all masked proofs can be unmasked.) Min value: 0, max value: 300.

3. *bogusMasks*: These occur when a non-empty answer includes values for hallucinated masks that do not appear anywhere in the given input. Min value: 0, max value: ∞ .
4. *missingMasks*: These occur when a non-empty answer is incomplete because the model fails to specify values for some of the masks in the input. Min value: 0, max value: μ .
5. *rulesToFormulas*: These occur when the model mistakenly lists a formula as the value of a mask that could only possibly assume a rule as its value. For instance, if the input proof contains a line such as

$$\text{MASK8 } \text{BY } \text{MASK12 } \text{on } \dots \quad (4)$$

and a model assigns MASK12: ($\sim A$), this constitutes a *rulesToFormulas* error, as it is patently obvious (by a purely syntactic/positional cue, the fact that MASK12 appears immediately before the **on** keyword) that MASK12 can only range over rules, not formulas. Min value: 0, max value: $\approx \mu$.

6. *formulasToRules*: These are similar to *rulesToFormulas* errors but err in the opposite direction: They occur when a model assigns a rule to a mask that can only receive formulas as values. Considering (4) again, an example would be a model that generated MASK8: left-and. Min value: 0, max value: $\approx \mu$.

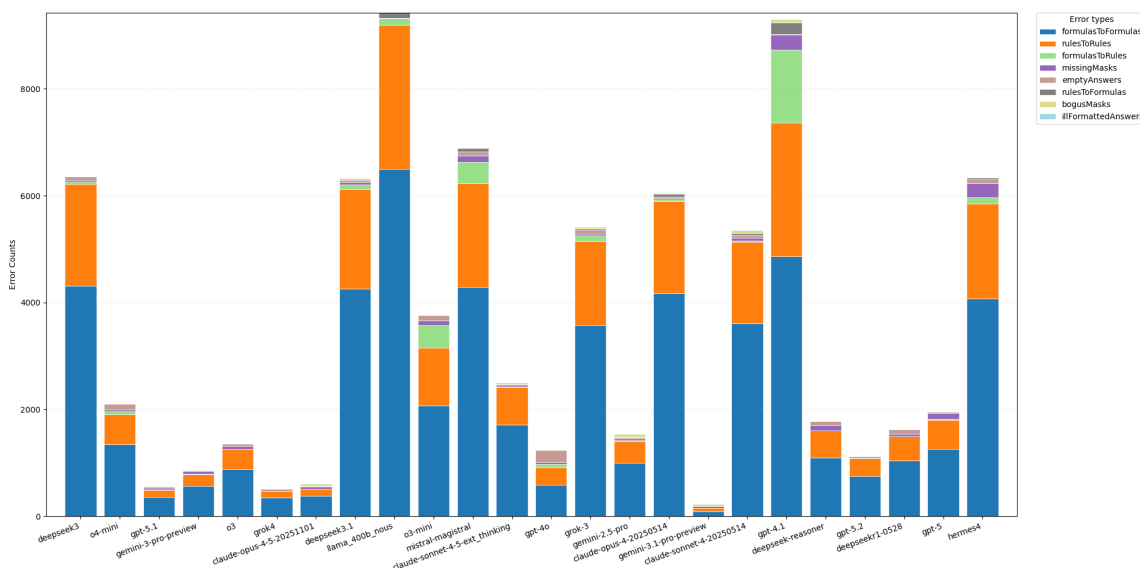


Figure 13. PL1-PM errors

These 6 categories represent *severe* errors. It is one thing to mix up one rule for another, but listing a rule as the value of a mask that can only range over formulas is a much more serious mistake. Likewise for the rest of the above error types, with the possible exception of *emptyAnswers*, a relatively less flagrant type of failure. There are two main types of errors that do not fall into any of these categories: *rulesToRules* errors (e.g., assigning mt to a mask whose correct value is mp), and *formulasToFormulas* errors, which occur when the wrong formula is listed as the value of a mask. These are more subtle errors, or at least less egregious, and round out our error typology for this task.

Fig. 13 plots the raw counts for these error types across all models. Note that we capped GPT-4.1's *formulasToRules* errors at 500 (its true value is 1,363) because using the full value would stretch the error-count axis so far that the lower counts of other error types would become difficult to read. We make the following observations:

- Empty answers very likely signal uncertainty (*I can't do this*) rather than random failures. As usual, these errors are common among OpenAI's mini models and earlier DeepSeek models, though we also see a remarkable spike for GPT-4o (222 empty answers out of a total of 300). These models are essentially using empty answers to broadcast uncertainty aversion: when an input crosses

some internal difficulty threshold (long proof or many masks), they emit an empty dictionary instead of venturing a guess.

- It is striking that even elite models make hard typing errors, of type *rulesToFormulas* or *formulasToRules* (e.g., GPT-5 makes 17 *formulasToRules* errors; the only model that makes zero such errors is Gemini 3.1). It's reasonable to conjecture that humans would very rarely make such errors, for the same reason they are unlikely to put a verb where an adjective is expected. Positional typing is a hard constraint for humans: after **assume** and before **BY** only formulas fit; immediately before **on** only rules fit; and so on. For LLMs, by contrast, these constraints appear to be "soft" or fluid. The fact that virtually all models make these mistakes, albeit to different degrees, means that models find it easier to verify typing constraints (as in **PL1-PC**) than to produce outputs that respect them. When generating, their decoders aren't gated by the relevant grammar/typing lattice and end up relying on token-level heuristics that occasionally slip across categories. This is the classic failure signature of generative brittleness (lack of a hard "type checker" in the loop). However, the frequency of these errors falls drastically among the top 4 models.
- *missingMasks* errors represent planning failures, or deficits in executive function. These are numerous even for top models (e.g., 109 for GPT-5, 53 for Gemini-3, 31 for GPT-5.2, 35 for GPT-5.1, and 34 for Gemini 3.1). Models here start filling and end up omitting keys. That's not an issue of misunderstanding NDL or making reasoning errors, it's an output orchestration issue—counting and enumerating all masks and persisting that plan through potentially long JSON outputs. It's the same phenomenon behind tool-call omissions in long chains.
- *bogusMasks* errors are relatively few but telling: when the list of required answers is long, some models hallucinate masks.
- Formatting errors are even rarer, but they are occasionally made, even by top models (2 such errors by o3). These errors obviously track prompt adherence and JSON discipline.

A Challenge Version of **PL1-PM**

In the remainder of this section we introduce **PL1-PM-c**, a challenge dataset containing 50 masked proofs having hundreds of lines (anywhere from 323 to 946 lines; the median is 400). As with the challenge version of **PL1-PC**, we only evaluate a small number of the most capable models. Each problem is evaluated with high reasoning settings, a high allowance reasoning tokens (100K), and a maximum wall-clock time of 30 minutes. If no output is produced within this time, the instance is marked as failed.

While this dataset involves longer proofs, it is less challenging than **PL1-PM** in a precise technical sense. Define the *mask density* of a masked proof D as the ratio $M(D)/L(D)$, where $M(D)$ is the number of masks⁵⁵ in D and $L(D)$ is the number of lines in D that may contain a mask, namely, all inference steps (applications of an inference rule) and all openings of conditional deductions (lines of the form **assume** · · ·). This simple statistic is the cleanest indication of how heavily masked a proof is. Low density means that most inference steps have clearly visible structure (i.e., their rule, arguments, and/or conclusion remain largely unobscured) and their masks are therefore locally forced with monotonic assignments. With high mask density, by contrast, some masks must be assigned values speculatively, forcing constraints only appear later, and incorrect assignments may survive for several steps (potentially dozens or even hundreds of steps). While the overall number of valid mask assignments might remain low (since mask values tend to be uniquely determined), the number of plausible partial assignments explored along the way can be large. In other words, high mask density increases entropy. The extreme case is fully masked inference steps like MASK51 **BY** MASK72 **on** MASK8, MASK12. Such steps have zero locally forced information and thus no way to commit without global reasoning. Again, this doesn't increase the number of solutions, only the depth of blind search required to discover one of the (relatively few) solutions.

The table below shows the distribution of mask density across both datasets.

⁵⁵ This is the same as the number of mask occurrences, since masks are unique.

	Mean	Median	Min	Max	Std	@5	@25	@50	@75	@90	@95
PL1-PM	1.72	1.64	0.32	4.0	0.49	1.11	1.4	1.64	1.95	2.24	2.67
PL1-PM_C	1.65	1.64	1.53	1.78	0.05	1.57	1.62	1.64	1.67	1.70	1.72

While the mean and median densities are very similar, **PL1-PM** has high variance (0.49) and quite a few outliers, e.g., a minimum density of only 0.32, which is essentially an unmasked proof. More importantly, it has a heavy right tail, with a significant proportion of heavily masked proofs (everything above the 75th percentile). In the challenge dataset, by contrast, the distribution of mask density is tightly concentrated, with a very small standard deviation (0.05). There are no heavily masked proofs on the right side of the median, and no outliers in either direction. The new dataset is thus near-constant density, uniformly enforcing consistent, moderate local difficulty.

Moreover, 16% of all inference steps across **PL1-PM** proofs are *completely* masked, whereas less than 1% of the inference steps in the challenge dataset are fully masked (0.1%, to be exact). At the macro level, 85% of proofs in **PL1-PM** have at least one inference step that is fully masked, whereas only 8% of the proofs in the challenge dataset have at least one such step. Recall that a fully masked inference step has no local syntactic anchor (no visible rule, conclusion, or arguments) and thus requires global constraint propagation to resolve; it cannot be solved incrementally. This means that **PL1-PM** problems regularly contain steps with zero local information, whereas challenge problems have them very rarely, marking an enormous reduction in local ambiguity.

A final notable difference is the masking of assumptions. Masking a hypothesis is one of the highest-entropy perturbations that can be made in a proof. The introduction of a hypothesis p in a line of the form `assume p` adds p to the assumption base, introduces a new logical scope for it, and determines the shape of the resulting conditional ($p \Rightarrow q$). If p is masked, the entire body of the conditional proof becomes underconstrained, as every inference inside that body may well depend on p , but we don't know what that is. And unless we manage to infer p with certainty, we cannot be sure of the conditional proof's conclusion either. In the challenge dataset, only 10% of conditional proofs have masked hypotheses, whereas in **PL1-PM**, 43% of conditional proofs (almost 1 in 2) have masked hypotheses.

These numbers show that the challenge dataset is not harder because its proofs are more masked or because they have fewer local hints. The opposite is true: the new dataset is technically easier because its proofs are less heavily masked and provide more local cues. The authors have tried solving a number of these longer problems by hand and have found it relatively straightforward to make consistent incremental progress; with sufficient patience and time, a human familiar with this proof style could finish such a problem. Yet this entropy reduction does not necessarily make the problem tractable at scale for models, as shown in the following results:

Model	Instrumented Accuracy	Strict Accuracy	Model	Instrumented Accuracy	Strict Accuracy
gpt-5.4	0.38 (± 0.13)	0.14 (± 0.1)	gemini-3.1-pro-preview	0.34 (± 0.13)	0.14 (± 0.1)
gpt-5.2	0.32 (± 0.13)	0.28 (± 0.12)	claude-opus-4-5-20251101	0.06 (± 0.07)	0.06 (± 0.07)

Table 8. PL1-PM-c instrumented and strict accuracies.

The difficulty has shifted from unmasking particular steps to maintaining a consistent global state across hundreds of steps.

Observe that while GPT-5.4 outperforms GPT-5.2 in terms of instrumental accuracy, its strict accuracy is much worse, half the strict accuracy of GPT-5.2 (though the Wilson confidence intervals are obviously wider here, since there are only 50 problems in the challenge dataset).

The reasoning traces of the models provide interesting insights. Here is part of what GPT-5.4 wrote while tackling item 9:

... In analyzing the subblock, I see that MASK188 must be biconditional ($D \Leftrightarrow B$) since the first line uses `left-iff` on MASK189. This means my assumption is set to ($D \Leftrightarrow B$). ... I need to explore how

this structure influences the rest of the proof and what implications arise from these assumptions. It's fascinating how these logical connections can shape the outcome! . . . The challenge is that B doesn't hold as expected. Could I assume MASK79 = B and label it? However, the line uses literal B and MASK80. Yet if I assume B, it holds in that context, even without naming it. If I set MASK79 to B, then it becomes an active assumption and holds. I see I overlooked that! . . .

Such traces must not be over-interpreted, since they may be narrative artifacts rather than genuine reflections of the model's internal computations. Nevertheless, they can still shed light on a number of fronts, e.g., on the style of reasoning that the model believes is appropriate or where it seems to experience higher amounts of uncertainty. The traces strongly indicate that these elite models are not simply pattern-matching; they are explicitly reasoning (or at any rate simulating a deliberation process) about the logical availability of conclusions, assumption scope, rule semantics, and possible mask assignments.

5. Proof Gaps

This dataset was obtained by randomly inserting holes (gaps) in a set of 200 correct proofs, written either by the Athena theorem prover or by a model. Unlike masks, which are locally focused on individual formulas and rules, gaps apply at the subproof level, i.e., a gap replaces an entire subproof, which could be either a single step (inference rule application), or a conditional subproof, or a *proof chain*: a sequence of consecutive subproofs. A proof usually receives more than one gap, in randomly selected subtrees. The total fractional amount of information elided by the gaps ranges from 0.3 to 0.99. In the latter extreme case, the entire proof is essentially replaced by a single gap, and the task then reduces to proof writing. The mean fractional total size of the gaps inserted in a proof is 0.67 of the original proof's size.

The objective is to fill the gaps by generating appropriate subproofs for the *gap identifiers*: GAP-1, GAP-2, . . . There are no distractors for this task, which we refer to as **PL1-GF**. The following is a simple example:

```

1 assert premise-1 := (C | (A & E))
2 assert premise-2 := (C ==> D)
3 assert premise-3 := (A <==> (E ==> D))
4 # Goal: (E ==> D)
5
6 {
7   left-iff on premise-3;
8   GAP-1;
9   assume E {
10    case1 := assume C {
11      mp on premise-2, C
12    };
13    case2 := assume (A & E) {
14      left-and on (A & E);
15      mp on (A ==> (E ==> D)), A;
16      GAP-2
17    };
18   GAP-3
19 }
20 }
```

Bearing the goal ($E \Rightarrow D$) in mind, it is clear from the structure of the overall proof skeleton, and particularly from the structure of the top-level conditional proof (the **assume** that opens on line 9 and closes on line 19), that this conditional proof establishes ($E \Rightarrow D$). The body proceeds by a case analysis of premise-1, the disjunction ($C \vee (A \wedge E)$). The case1 subproof establishes that C implies D , thus case2 must establish that $(A \wedge E)$ implies D . Therefore, GAP-2 must derive D from the assumption base at the end of line 15, which includes the conditional ($E \Rightarrow D$) (derived on line 15) and the case-2 hypothesis ($A \wedge E$) (as GAP-2 is inside the scope of that hypothesis). So this gap is easy to fill: first detach E from the assumption ($A \wedge E$) and then use modus ponens on ($E \Rightarrow D$) and E to infer D :

GAP-2: E **BY** right-and on (A & E); D **BY** mp on (E ==> D), E

GAP-3 is a simple rule application: cases on premise-1, $(C \implies D)$, $((A \ \& \ E) \implies D)$.

Note that GAP-1 does not make any contribution to the proof. It is there only because the LLM that generated the initial proof included a redundant step at that point, namely, the application of right-iff on premise-3. Any replacement for that gap is given full credit as long as the entire proof obtained by filling the gaps as specified by the model correctly derives the goal from the premises. This is the only criterion we use for correctness.

Statistics on the problem distribution for this task slice are shown in Fig. 14. The problems are quite simple and roughly similar in their complexity to masking problems, except that conclusions are notably smaller (the median number of nodes is 3). The statistics below show that the lengths of these proofs are roughly in the same range as those in PL1-PM, but on average these proofs are about 25% longer (34.5 lines vs 27).

	Mean	Median	Min	Max	Std	@5	@25	@50	@75	@90	@95
Proof length	34.5	39.5	3	64	15.89	11.95	20.0	39.5	49.0	53.0	53.0

Accuracies for all models, obtained both from the instrumented and the regular proof checkers, are shown in Table 9.

The discriminability index (DI) of PL1-GF is 37.5%, meaning (roughly) that any two of these models differ in accuracy by about 37.5 percentage points—again a very large spread. This task strongly separates some elite models as well, e.g., the delta between o3 and Gemini 3 Pro Preview (as well as the GPT-5-series models, Opus 4.5 and Gemini 3.1) is highly statistically significant, while the delta between, say, Grok-4 and Claude Sonnet 4.5 with extended thinking is statistically significant.⁵⁶

We treat blank (empty-string) responses after a timeout budget of at least 30 minutes as genuine task failures (non-responses) that count as incorrect outcomes in our accuracy computation. By contrast, if there is a genuine API call error we do not include that data point in the accuracy denominator, and in such cases we try to repeat the call from scratch;⁵⁷ see Appendix C for additional details. For transparency, we also report non-response rates: Grok-4 abstained on 11/200 problems (5.5%), Gemini 3 Pro Preview on 7 problems (3%), Mistral Magistral on 7 problems (3%), and o3 on 1 problem (0.5%). All other models returned non-empty responses for all 200 problems.

Formatting issues were rife for some models, particularly for the two Claude 4 models (Opus and Sonnet), which frequently failed to return a valid gap assignment, or indeed any gap assignment, instead emitting a random fragment of NDL code.⁵⁸ This was a stable behavioral pattern, confirmed by recomputing results multiple times. Other models made formatting errors of different types, some

⁵⁶ We used McNemar’s test to check whether these pairwise deltas could be due to random variation. We call the p -value of a particular test result “statistically significant” if it’s below the usual 0.05 threshold and “highly statistically significant” if it’s below the 0.01 threshold. In the case of Gemini 3 Pro Preview and o3, for example, out of 38 problems where the models disagreed, Gemini was correct on 31 and o3 on only 7, which corresponds to a p -value of 0.000116. This means the likelihood that such a large advantage occurred by chance is less than 0.1%, and thus the difference in performance between these two models is very statistically robust.

⁵⁷ In the final results there is only one API call error left, for Mistral Magistral on item 135.

⁵⁸ Claude 4 Opus did this for 135 out of 200 problems (67.5% of all inputs) and Claude 4 Sonnet for 119 problems (59.5%).

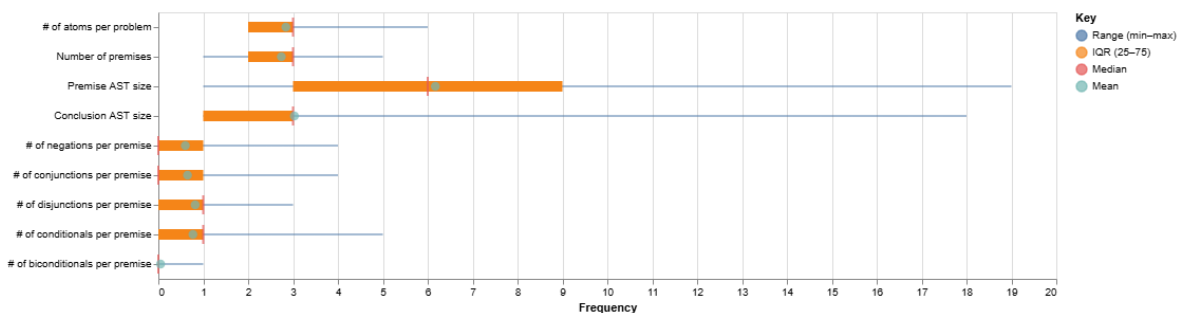


Figure 14. PL1-GF problem statistics

Model	Instrumented Accuracy	Strict Accuracy	Model	Instrumented Accuracy	Strict Accuracy
claude-opus-4-5-20251101	0.94 (± 0.03)	0.88 (± 0.05)	gemin-3.1-pro-preview	0.92 (± 0.04)	0.89 (± 0.04)
gpt-5.2	0.93 (± 0.04)	0.84 (± 0.05)	gpt-5	0.90 (± 0.04)	0.79 (± 0.06)
gemin-3.0-pro-preview	0.89 (± 0.04)	0.77 (± 0.06)	gpt-5.1	0.89 (± 0.04)	0.70 (± 0.06)
claude-sonnet-4.5-ext-thinking	0.80 (± 0.06)	0.60 (± 0.07)	o3	0.77 (± 0.06)	0.56 (± 0.07)
gemin-2.5-pro	0.75 (± 0.06)	0.49 (± 0.07)	grok-4	0.69 (± 0.06)	0.55 (± 0.07)
deepseek-reasoner	0.61 (± 0.07)	0.31 (± 0.06)	o4-mini	0.45 (± 0.07)	0.37 (± 0.07)
deepseek-r1-0528	0.46 (± 0.07)	0.24 (± 0.06)	grok-3	0.38 (± 0.07)	0.32 (± 0.06)
gpt-4.1	0.29 (± 0.06)	0.24 (± 0.06)	deepseek-v3	0.30 (± 0.06)	0.21 (± 0.06)
deepseek-v3.1	0.30 (± 0.06)	0.18 (± 0.05)	mistral-magistral	0.24 (± 0.06)	0.19 (± 0.05)
o3-mini	0.23 (± 0.06)	0.17 (± 0.05)	claude-sonnet-4-20250514	0.16 (± 0.05)	0.15 (± 0.05)
claude-opus-4-20250514	0.14 (± 0.05)	0.13 (± 0.05)	hermes-4	0.10 (± 0.04)	0.07 (± 0.04)
llama-3.1-405b	0.07 (± 0.04)	0.03 (± 0.03)	gpt-4o	0.01 (± 0.02)	0.01 (± 0.02)

Table 9. PL1-GF accuracies, using both the instrumented and the regular proof checker.

more exotic and blatant than others. Unlike other PROOFGRID tasks, the **PL1-GF** prompt does not require JSON output, instead instructing the models to simply think through the problem and then write their solutions at the very end of their output, in the following simple form:

```
GAP-i:
<subproof for GAP-i>
GAP-j:
<subproof for GAP-j>
...

```

Only the latest generation of frontier models adhered faithfully to these instructions. All other models deviated, typically in predictable ways that could be programmatically handled.⁵⁹

```
**GAP-5** :
```nd1
assume (C & D) {
...
}

GAP-8 :
...
...
...

```

But some less common formatting issues resulted in outputs that could not be easily salvaged and had to be counted as incorrect. For example, in a few cases (most notably involving o3-mini), the NDL proof fragments assigned to some gaps contained references to other gaps. Another example: occasionally the model's response would only contain a COT trace in natural language, without ever culminating in a gap assignment. Other times there would be additional commentary inserted after the supposedly final gap assignment; other times the model would insert a gratuitous delimiter (such as a long dashed line) in between the different proof fragments; and so on.

A more common error was incomplete gap assignments: properly formatted outputs that nevertheless neglected to assign values to some of the input gaps. Here Claude 4 Opus and Sonnet lead the pack with 144 and 130 respectively (due to the malformed responses mentioned earlier), followed by GPT-4o (30), Mistral Magistral (21), Grok-4 (12), o4-mini (9), DeepSeek-V3 (8), Gemini- 2.5-Pro (5), GPT-4.1 (4), Grok-3 (3), the Nous-tuned version of Llama-400b (1), the extended-thinking version

<sup>59</sup> For example, many models insisted on writing the GAP-i headers in boldfaced markdown, writing, for example, **\*\*GAP-8\*\***: or **\*\*\*GAP-8\*\*\***: instead of GAP-8:; or in different (and seemingly arbitrary) markdown heading levels; sometimes they would insert the colon before the asterisks, e.g. writing **\*\*GAP-8:\*\***; and so on. And many models enclosed the generated NDL fragments inside triple backticks (another common markdown convention, for marking code blocks), e.g. writing something like:



coefficient between accuracy and normalized error-repair density (per 100 lines) is -0.82. DeepSeek R1-0528 is interesting. It has a lot of error repairs but manages to come close to 50%, slightly outperforming o4-mini. However, the vast majority of these errors are syntactic, and most of them are due to dropping outer parentheses from complex formulas. In other words, most of the errors that R1-0528 makes and the instrumented checker fixes are ultra-local and low-impact systematic deviations that violate strict formalism without breaking logical coherence. These are exactly the sort of lapses that the instrumented checker was designed to tolerate.

As shown in Table 9, and as was also the case for **PL1-PW**, the instrumented checker raises all model accuracies significantly, except for the bottom tier of the 8 worst-performing models, where improvements are marginal. As explained earlier, that was the intention behind the instrumented proof checker: to try to decouple formal precision from logical competence by overlooking minor errors. Nevertheless, the overall correlation between repair rate and accuracy remains strongly negative, as shown by the scatterplot and the correlation coefficient. This is not contradictory. At an intra-model level of analysis, we see that instrumentation unlocks latent reasoning ability that would otherwise be obscured by procedural slips. At an inter-model or global level of analysis, across models, a high rate of procedural error repairs indicates weaker executive control, so models that generate formally correct proofs also attain higher reasoning accuracy. In other words, formal discipline and semantic insight appear to co-develop in LLMs. This suggests that reasoning and instruction-following share an underlying architectural or representational substrate. And it also indicates that as models scale and align better, they improve not just in raw inference ability but also in procedural compliance, a skill that is essential both in logic and in coding.

Models tend to do better on proof writing than on gap filling (compare Tables 4 and 9). Sometimes the difference is dramatic, e.g., a 19-point gap in the case of Gemini 2.5 Pro. This might seem counterintuitive. Gap filling introduces a proof-synthesis spectrum, where on one end we have a tiny number of small gaps that do not make any significant synthesis demands, while on the other end we have very large gaps that essentially require models to write the entire proof. It would seem that the problem would get harder as we move along this continuum.

However, gap-filling introduces constraints that are not present in proof writing. In **PL1-PW**, a model has the freedom to choose any proof strategy it likes, to use “favorite” inference patterns, discharge assumptions in whatever order it deems appropriate, and so on. There is no need to respect and follow a pre-existing plan. In **PL1-GF**, by contrast, the input proof skeleton forces a specific problem decomposition with a particular sequence of subgoals and hardwired local plans, which the model has to reverse-engineer, e.g., by deducing why certain surrounding lines are there. It is tackling a planning-under-constraints problem rather than a plain proof-writing problem. While plain proof writing in propositional logic can be viewed as a classic forward planning problem, gap filling imposes a hidden inverse search problem, in the technical sense of inverse Bayesian planning, where the goal is to abductively infer the intentions of the agent who wrote the initial proof by “inverting a model” of how intentions cause proof-writing behavior (Baker et al. 2009).<sup>60</sup> Depending on its exact formulation, inverse planning is at least as computationally hard as forward planning. Gap-filling also demands stronger meta-reasoning skills, since it requires explicit deliberation about the given proof skeleton and constructs like assumption scope. (All inverse tasks generally require reasoning about another agent’s reasoning processes, thus exercising a sort of theory-of-mind skillset.) These considerations could help to explain why, for example, Grok-4 goes from 85% accuracy in **PL1-PW** to 69% in **PL1-GF**. In general, the diversity of tasks in PROOFGRID allows for richer profiling that separates models along different axes of ability.

Next, we analyze the first gap-filling errors made by models, i.e., those errors that the instrumented interpreter does not fix. These are divided into the same six groups introduced in the **PL1-PW** error analysis. Their distribution is shown in Fig. 16. As in other tasks, three main cohorts emerge from this

<sup>60</sup> In classical planning this is related to *plan recognition*, also known as “inverse planning,” where one observes a sequence of actions and must infer the goals that gave rise to them (Ramirez and Geffner 2010).

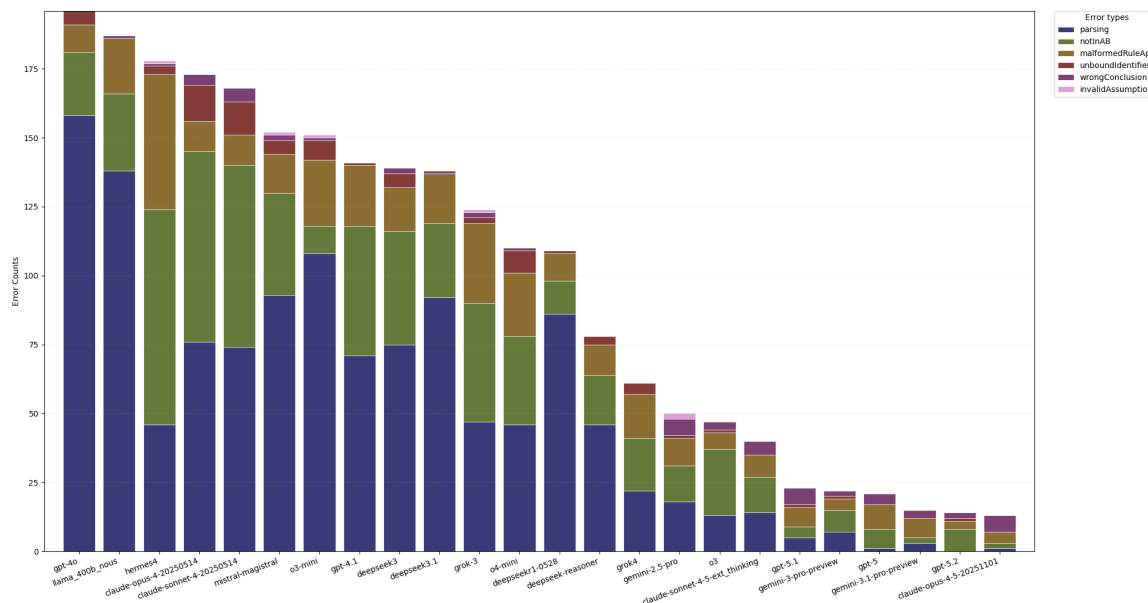


Figure 16. PL1-GF distribution of first errors.

picture:

- The surface-fragility regime of weaker models like GPT-4o and Llama-400b, whose error profiles are dominated by parsing/syntax (blue) failures. These models fail before their reasoning can be scrutinized. Instrumentation can't help because the emitted fragments deviate from proper syntax in highly irregular ways that cannot be programmatically detected. As two simple examples, instead of writing A **BY** left-and on **on**  $\dots$ , a model in this cohort might write A INFERRED BY left-and on  $\dots$ , or instead of (A&B) they might write A AND ALSO B. We have anecdotally observed that even if these grammar issues were somehow resolved, the generated fragments would still fail due to downstream structural and logical errors. In other words, parsing errors and logical failures are entangled phenomena for weaker models, not independent pathologies.
- The mixed-failure regime of mid-tier models, where parsing errors range from  $\approx 30\%$  to  $60\%$  of the total, the rest coming from logical errors (citing formulas that are not in the assumption base), misapplied inference rules, and references to unbound identifiers.
- The top-tier models, whose errors are predominantly semantic/logical or structural. GPT-5.2 stands out with *zero* parsing errors, an indication of its exceptional instruction-following ability.

We close by examining how gap-filling performance scales with input size. Fig. 17 plots the accuracy of the top 12 models as a function of the input proof length, while Fig. 18 does the same for the bottom 12 models. These curves show that the accuracy of all models falls as the number of proof lines in the input increases. However, while weaker models exhibit a sharp and steadily monotonic performance collapse, with many hitting near-zero accuracy around 40 lines, the declines of the stronger models are much more gradual and less catastrophic.<sup>61</sup>

<sup>61</sup> Some models either plateau or show a small uptick at the tail end. This is likely due to a gap-fraction confound (by “gap fraction” we mean the proportion of the original proof that was replaced by gaps). Generally, once we control for proof length, the problem gets harder as the gap fraction increases. Likewise, if the gap fraction is fixed, the problem gets harder as proof length increases. However, because PL1-GF is a small dataset (200 data points) and not perfectly stratified across both features, longer proofs tend to have smaller gap fractions, while shorter proofs have higher gap fractions. This imbalance explains why a plot of accuracy vs gap fraction would likewise show a declining sensitivity curve with a small rebound at the tail end: because of a confounding effect, this time (symmetrically) due to proof length. In both cases, the apparent slight improvement at the tail happens because the other variable is easier in the corresponding region.

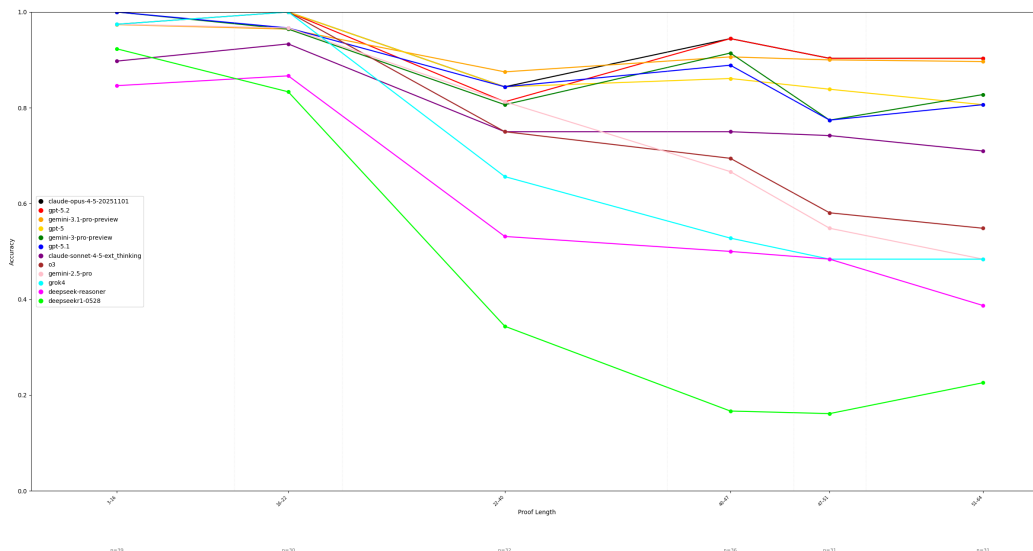


Figure 17. PL1-GF accuracy vs gapped proof length for the top 12 models

A Challenge Version of PL1-GF

In this section we discuss a more challenging version of this task, **PL1-GF-c**, with 50 problems featuring longer proofs. The median proof length is 399, and 90% of the proofs have fewer than 510 steps. The maximum length is 946. The median number of gaps per proof is 45, and the maximum is 100.

Although these proofs are longer, the gaps are easier to fill for a number of reasons. The original proofs for this dataset were generated by a simple automated theorem prover, and as a result they have a great deal of structure and a number of recurring patterns. In addition, gap completions tend to be locally short and semantically anchored. (For instance, among the set of correct answers generated by GPT-5.4, the median gap completion is 1, and the average is 2.6; 90% of all gap completions have fewer than 6 lines.) As an example, below is the top portion of a proof from **PL1-GF-c**.

```

1 assert premise-1 := ((D | C) & (A ==> ~D))
2 assert premise-2 := ((~ ((~ (A | C) ==> D)) | ((~ (~B | C)) <==> A))
3 assert premise-3 := (~C | (A & B & ~E) | (A & D & ~E) | (E & ~B & ~D) | (~A & ~B & ~D))
4 # Derive (~A & ~B)
5 {
6 p0 := (D | C) BY left-and on ((D | C) & (A ==> (~ D)));
7 p1 := (A ==> (~ D)) BY right-and on ((D | C) & (A ==> (~ D)));
8 ((~ A) | (~ D)) BY cond-def on (A ==> (~ D));
9 assume p2 := D {
10 assume p3 := (E & ((~ B) & (~ D))) {
11 ((~ B) & (~ D)) BY right-and on p3;
12 (~ D) BY right-and on ((~ B) & (~ D));

```

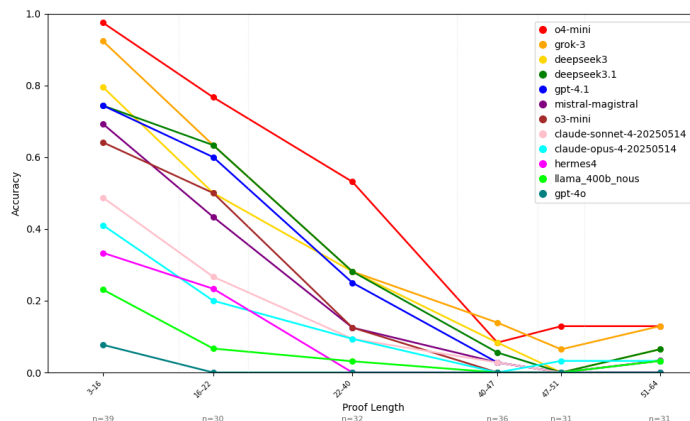


Figure 18. PL1-GF accuracy vs gapped proof length for the bottom 12 models.

```

13 GAP-13
14 };
15 assume ((~ A) & ((~ B) & (~ D))) {
16 GAP-10;
17 p3 := (~ D) BY right-and on ((~ B) & (~ D));
18 ((~ A) & (~ B)) BY from-complements on ((~ A) & (~ B)), p2, (~ D)
19 };
20 assume ((E & ((~ B) & (~ D))) | ((~ A) & ((~ B) & (~ D)))) {
21 ((~ A) & (~ B)) BY cases on ((E & ((~ B) & (~ D))) | ((~ A) & ((~ B) & (~ D)))),
22 ((E & ((~ B) & (~ D))) ==> ((~ A) & (~ B))),
23 (((~ A) & ((~ B) & (~ D))) ==> ((~ A) & (~ B)))
24 };
25 ...

```

It is immediately clear from the structure of the case analysis on lines 22–24 that the first inner conditional proof (lines 11–15) must produce the conditional  $(\neg E \wedge \neg B \wedge \neg D \Rightarrow \neg A \wedge \neg B)$  and hence GAP-13 must produce  $(\neg A \wedge \neg B)$ , which can easily be done with an application of from-complements on  $D$  and  $(\neg D)$ , exactly as is done on line 19. Likewise, GAP-10 is easily seen to be the counterpart of line 12. We thus have:

```

GAP-10: (~B & ~D) BY right-and on (~A & (~B & ~D))
GAP-13: (~A & ~B) BY from-complements on (~A & ~B), D, (~ D)

```

The remaining gaps can be filled in a similar way.

Results for the 4 currently strongest models are shown in Table 11.

Model	Instrumented Accuracy	Strict Accuracy	Model	Instrumented Accuracy	Strict Accuracy
gpt-5.4	0.51 ( $\pm 0.13$ )	0.51 ( $\pm 0.13$ )	gpt-5.2	0.35 ( $\pm 0.13$ )	0.31 ( $\pm 0.12$ )
gemini-3.1-pro-preview	0.25 ( $\pm 0.12$ )	0.24 ( $\pm 0.11$ )	claude-opus-4-5-20251101	0.0 ( $\pm 0.04$ )	0.0 ( $\pm 0.04$ )

**Table 11.** PL1-GF-c instrumented and strict accuracies.

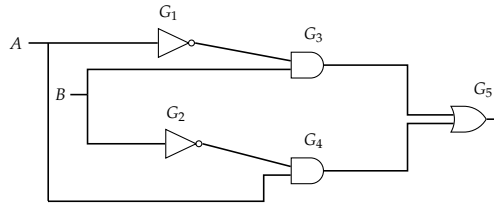
The score spread across these 4 powerful models is remarkably wide. The mean numbers of proof lines written across all 50 problems are 132 for GPT-5.4, 145 for GPT-5.2, 134 for Gemini, and 46 for Claude Opus, which fails to return meaningful gap completions for about half of the problems.

The point-biserial correlation coefficients between original proof lengths and outcome success indicate that, for these models, proof length is not correlated with task success. These coefficients are -0.15 for Gemini 3.1, -0.04 for GPT-5.4, and 0.09 for GPT-5.2 (the correlation is meaningless for Claude). This suggests that, for elite models, raw proof length is not the operative difficulty variable in this task. We conjecture that the dominant source of difficulty is average gap size (or, more generally, the amount of omitted proof material per gap), but leave further analysis to future work.

## 6. PL<sub>2</sub>

As we pointed out earlier, PL<sub>1</sub> problems are easy because they involve few atoms and randomly constructed formulas of low structural complexity. The PL<sub>2</sub> division contains richer and more complex problems with more atoms. Nevertheless, problems are still relatively short, with about 10 premises on average, and premises also tend to be short, with fewer than 10 nodes on average. We start by illustrating the generation of PL<sub>2</sub> problems with a concrete example, after which we describe a general algorithm that can take an arbitrary formula as input and construct a PL<sub>2</sub> problem from it. For reasons that will become clear shortly, PL<sub>2</sub> problems have a strong abductive flavor, despite their deductive formulation. Indeed, the algorithm we describe below could also be used to generate proper abductive-inference problems, where the desired output is a minimal hypothesis that explains a set of observations.

Consider the formula  $p = ((\neg A \wedge B) \vee (\neg B \wedge A))$ . We can view  $p$  as the following circuit:



The atoms  $A$  and  $B$  represent the inputs to the circuit, while  $G_1, \dots, G_5$  represent the outputs of the five gates. The output of the whole circuit is  $G_5$ , produced by the fifth gate. We introduce 5 fresh atoms  $N_i$  for  $i = 1, \dots, 5$ , where  $N_i$  means that the  $i^{\text{th}}$  gate,  $G_i$ , is functioning normally:

$$N_1 \Rightarrow (G_1 \Leftrightarrow \neg A) \quad (5)$$

$$N_2 \Rightarrow (G_2 \Leftrightarrow \neg B) \quad (6)$$

$$N_3 \Rightarrow (G_3 \Leftrightarrow (G_1 \wedge B)) \quad (7)$$

$$N_4 \Rightarrow (G_4 \Leftrightarrow (G_2 \wedge A)) \quad (8)$$

$$N_5 \Rightarrow (G_5 \Leftrightarrow (G_3 \vee G_4)) \quad (9)$$

We'll use the letter  $\mathcal{T}$  to refer to the set of formulas (5)–(9). Using these formulas as premises, we can prove that if all gates are functioning normally, then  $G_5$  computes the exclusive-or function over  $A$  and  $B$ . More precisely:

$$\mathcal{T} \cup \{N_1, N_2, N_3, N_4, N_5\} \models (G_5 \Leftrightarrow ((\neg A \wedge B) \vee (\neg B \wedge A))),$$

and we could write a proof that derives the conclusion on the right-hand side of the turnstile by taking the formulas on the left-hand side as premises. Perhaps more interestingly, assuming again  $\mathcal{T}$ , and assuming  $p$ , we can prove that if  $G_5$  does *not* hold then at least one of the gates must be malfunctioning:

$$\mathcal{T} \cup \{p, \neg G_5\} \models \neg N_1 \vee \neg N_2 \vee \neg N_3 \vee \neg N_4 \vee \neg N_5. \quad (10)$$

This says that, given  $\mathcal{T}$ , if the inputs to the circuit (the atoms  $A$  and  $B$ ) satisfy  $p$  and yet the output bit  $G_5$  is 0, then at least one of the gates is not operating normally.<sup>62</sup>

While this discussion is couched in deductive terminology, the underlying reasoning is inherently abductive, as this is a classical *model-based diagnosis* problem. The normal-operation definitions form our *theory*  $\mathcal{T}$ , while  $p$  and  $\neg G_5$  serve as our *observations*  $O$ , reporting that the output bit  $G_5$  is 0 and that the inputs  $A$  and  $B$  satisfy  $p$ .<sup>63</sup> It should be clear that  $\mathcal{T} \cup O \cup \{N_1, \dots, N_5\}$  is inconsistent, and the obvious way to restore consistency is to reject at least one normality hypothesis, which is precisely what (10) does.

We now describe a general algorithm  $Ab$  that takes any propositional formula  $p$  (whose AST is denoted by  $ast(p)$ ) and produces a sequent of the form

$$\mathcal{T} \cup \{p, \neg G_{root}\} \vdash \bigvee_{v \in nodes(ast(p))} \neg N_v, \quad (11)$$

<sup>62</sup> Note that, given  $\mathcal{T}$ , the formulas  $p$  and  $\neg G_5$  by themselves are not inconsistent. This is because the “theory”  $\mathcal{T}$  does not allow us to conclude  $(G_5 \Leftrightarrow p)$  unconditionally;  $p$  and  $G_5$  become equivalent (and hence  $p$  and  $\neg G_5$  become inconsistent) only if we also assume that all gates are operating normally.

<sup>63</sup> Typically, “observations” in a particular test case are *concrete*, e.g., individual sensor readings for inputs and outputs, such as  $A = 0, B = 1, G_5 = 0$ . These are naturally represented by literals (atoms or their negations). This makes our use of a complex formula like  $p$  as an observation somewhat unorthodox, as  $p$  represents not just one particular input state but multiple states. Essentially,  $p$  serves as a constraint over the atoms of the input formula. Its use as a premise means that the atoms are in some (any) assignment where their XOR is 1, but we don't bother to specify a particular assignment.

where  $\mathcal{T}$  is a set of guarded definitional clauses. The intuition is that if all gates are normal, then the circuit output  $G_{root}$  must equal  $p$ . Therefore, if  $p$  holds but  $G_{root}$  is observed false, at least one gate must be abnormal.

**Step 1** Parse the input formula  $p$  into  $ast(p)$ . Each internal node contains a propositional connective, while leaves are atoms.

**Step 2** For every AST node  $v$ , introduce two fresh atoms:

- $G_v$ : represents the output wire of the gate computing the subformula at  $v$ .
- $N_v$ : represents the statement that the gate at  $v$  is normal.

**Step 3** Add guarded definitions. For each internal AST node  $v$  with operator  $op_v$  and children  $u_1, \dots, u_k$ , add the guarded equivalence

$$N_v \Rightarrow (G_v \Leftrightarrow op_v(G_{u_1}, \dots, G_{u_k})).$$

Collect all such guarded definitions into the set  $\mathcal{T}$ .

**Step 4** Transform the guarded definitions. With probability 0.5, each conditional  $p_1 \Rightarrow p_2$  in  $\mathcal{T}$  becomes either the disjunction  $\neg p_1 \vee p_2$  or the contrapositive  $\neg p_2 \Rightarrow \neg p_1$ .

**Step 5** Identify the root. Let  $root$  denote the root of  $ast(p)$ . Its gate variable is  $G_{root}$ .

**Step 6** Add the observations. Specifically, we add two observation formulas:

- $p$  (the specification is satisfied for the current input assignment).
- $\neg G_{root}$  (the actual output wire is observed false).

**Step 7** Formulate the conclusion and produce the output. Define the disjunction

$$C = \bigvee_{v \in nodes(p)} \neg N_v.$$

The target output sequent is then:

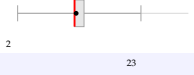
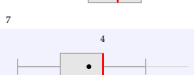
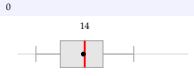
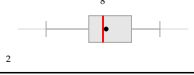
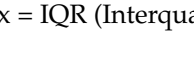
$$\mathcal{T} \cup \{p, \neg G_{root}\} \vdash C. \quad (12)$$

If all  $N_v$  are true, every clause in  $\mathcal{T}$  reduces to a Tseitin-style definitional equality.<sup>64</sup> By induction on the AST, this yields  $G_{root} \Leftrightarrow p$ , which is inconsistent with the observations  $\{p, \neg G_{root}\}$ . Therefore, not all  $N_v$  can be true, i.e., at least one  $\neg N_v$  holds. This is the output entailment problem.

**PL2-PW** contains 200 problems that were obtained from **PL<sub>1</sub>** problems as follows. Given a problem with  $n$  premises  $p_1, \dots, p_n$  and goal  $q$ , we sort the list of  $n + 1$  formulas  $p_1, \dots, p_n, q$  in order of decreasing length and choose a random initial segment  $S$  of the sorted list that has at least 2 elements. We then negate each element of  $S$  with probability 0.5 and join the results into a single complex formula  $p$  via random propositional connectives. For example, if  $S$  initially contains 3 formulas  $p_1, p_2, p_3$ , then these formulas might be joined into  $p = ((p_1 \wedge \neg p_2) \vee p_3)$ . The proof problem is then  $Ab(p)$ , i.e., the result of applying the algorithm  $Ab$  described above to the formula  $p$ .

All **PL2-PW** problems thus share a common logical template, namely (12). While the input  $p$  varies widely, the overall construction still imposes a great deal of schematic uniformity on the resulting problems. This homogeneity limits diversity and makes **PL2-PW** less suitable for gauging general proof-writing ability. Rather, this task measures generative performance on a family of structurally related, abductively flavored reasoning problems. Consequently, it is plausible that once a model internalizes the strategy induced by the problem generator (in our case, how to reason backward from

<sup>64</sup> The atoms  $G_v$  are exactly the Tseitin variables one would introduce in a Tseitin CNF encoding of  $p$ . The difference here is that each definitional equivalence is guarded by a normality atom  $N_v$ . Dropping the guards yields the usual Tseitin encoding.

Quantity	Mean	SD	Median	Min	Max	Spark
# of premises per problem	9.41	1.94	10	4	13	
# of atoms per problem	17.55	4.0	18	6	25	
Premise AST size	8.17	3.00	8	2	20	
Goal AST size	21.23	5.82	23	5	32	
# of negations per problem	23.02	6.54	24	7	36	
# of conjunctions per problem	3.34	1.96	4	0	8	
# of disjunctions per problem	13.77	5.0	14	3	31	
# of conditionals per problem	7.03	2.95	7	0	17	
# of biconditionals per problem	8.22	2.32	8	2	14	

**Table 12.** Data distribution summaries for the 200 problems in **PW-3**. Box = IQR (Interquartile Range); red line = median; dot = mean; whiskers = 5–95th.

the disjunction over the  $\neg N_v$  literals to the structure of  $p$ 's AST), it becomes capable of transfer across instances. Essentially, the concern is that if a model figures out how to solve one problem of this form, it may become able to solve all of them, modulo size constraints.

That said, template-driven problem generators are a common theme in evaluation. Many classic benchmarks (in maze and puzzle solving in general, in circuit evaluation, in SAT families, and elsewhere) rely on uniform or parametric generators—and for good reason, since these allow for controlled scaling, ablations, and clean attribution of failure modes. More importantly, structure discovery is not free; figuring out the underlying algorithm is itself the core difficulty. If a system reliably discovers and executes the correct reasoning chain, that's a real capability, not a loophole. In the case of **PL2-PW**, most models fail to do so despite the shared structure of the problem instances, indicating that the task is far from trivialized by the underlying semantic homogeneity. This is consistent with other benchmark tasks, both within **PROOFGRID** and elsewhere. In **PL<sub>3</sub>**, for instance, all problems from the same family (e.g., all graph coloring or subset cardinality problems) are structurally similar, but several families remain challenging nevertheless. Even if a model somehow learns “the general idea” behind a problem, this still doesn't ensure that it will be able to solve any given instance, since its performance will still be limited by its planning horizon, its working memory, error accumulation, and combinatorial explosion.

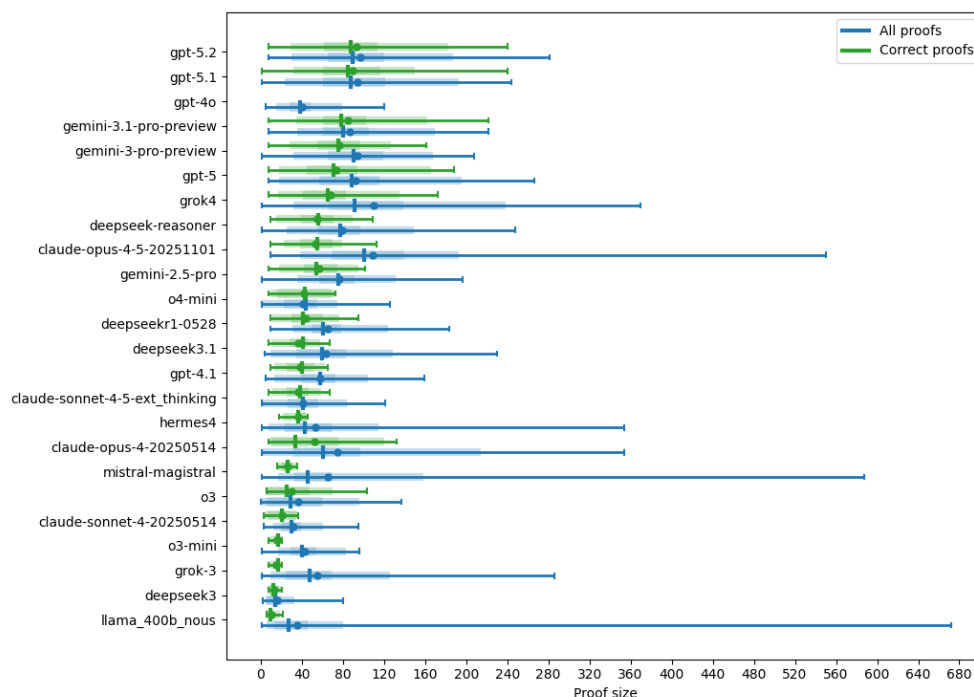
Table 12 summarizes some key characteristics of these 200 problems. **PL<sub>2</sub>** problems have more premises (the median is 10), but premises tend to be short. This is to be expected, given the algorithm we just described. We have more premises because a new formula is introduced for each internal node of the starting formula  $p$ . And because these new formulas are guarded conditional clauses that are frequently of the form  $(N_u \Rightarrow (G_v \Leftrightarrow G_r \text{ op } G_t))$ , their size is often exactly 7, so premises tend to be short (the inclusion of the starting formula  $p$  nudges the average size slightly higher). The number of atoms per **PL2-PW** problem is considerably higher, with a median of 18, since two new atoms ( $N_v$  and  $G_v$ ) are introduced for each node  $v$  in  $ast(p)$ . Goals here are also more complex, since they are disjunctions of negated normality atoms. In addition, recall that the *Ab* algorithm randomly negates each formula in the initial segment  $S$  described above with probability 0.5; and every initial guarded

conditional in  $\mathcal{T}$  is turned into a formula with at least one negation sign (Step 4 of the algorithm). As a result, these problems are negation-heavy, with the median number of negations at 24. Note that each premise has fewer than 2 negation signs on average (1.66, to be exact), but these add up over the entire problem simply because there are more premises. The main drivers of complexity in this task slice are the higher numbers of atoms and negations.

We close this section with a brief description of how the *Ab* algorithm could be modified to generate proper abductive-inference problems. The target conclusion of problem (11) can be viewed as an explanatory hypothesis or a diagnosis, but it is a rather uninformative one insofar as it essentially says that “something is wrong,” i.e., that at least one gate is malfunctioning. This is because our chief observation pertains to the top-level (“root”) node, which clearly depends for its function on every other node, and we have no information about the outputs of any other nodes. But the algorithm could be easily tweaked to generate proper abductive-inference problems targeting much more informative explanations, e.g., by randomly choosing a concrete interpretation  $I$  (assignment of truth values to  $p$ 's atoms) as a concrete test case and an internal node (“gate”)  $v$  in  $ast(p)$ . We would then compute the expected value of  $G_v$  under  $I$  (1 or 0) and take as a premise the literal that asserts the opposite of that value. The target hypothesis will then be that one of  $v$ 's predecessor nodes must be malfunctioning. For example, choosing  $I = \{A \mapsto 1, B \mapsto 0\}$  and  $v$  as the node corresponding to  $G_4$ , we see that if everything is working properly then  $G_4$  should hold (its value should be 1). We thus take the opposite,  $\neg G_4$ , as a premise, along with  $(A \wedge \neg B)$  (the formula representing  $I$ ) and  $\mathcal{T}$ . With this information as background, the ground-truth hypothesis would be  $(\neg N_2 \vee \neg N_4)$ . The target diagnosis could be sharpened further if we provide additional information about specific gates, e.g.,  $N_2$  or  $G_2$ .

### 6.1. Results

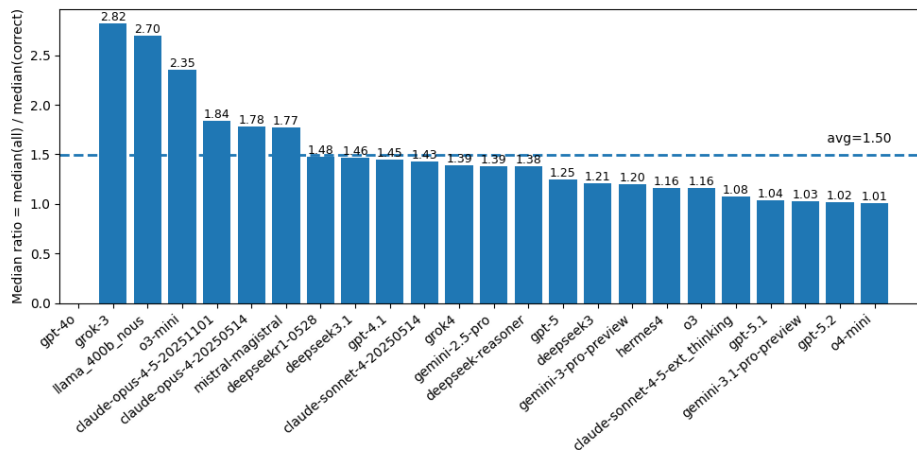
Statistics on the lengths of the proofs written by all models for the **PL2-PW** proofs are shown in Fig. 19. For each model  $M$ , the figure shows the distribution of length over the set of all proofs written



**Figure 19.** PL2-PW proof-size distributions per model: all proofs (blue) vs correct proofs (green); min–max line, 5–95% band, IQR box, median tick, mean dot.

by  $M$  (blue line) as well as the length distribution over the set of correct proofs written by  $M$ . Note that average proof sizes for both sets are about an order of magnitude higher than those in **PL1-PW**.

The green (correct-proofs-only) distributions are narrower and shifted left compared to the blue (all-proofs) distributions, showing that correct proofs tend to be shorter than the full set of proofs that a model generates. Incorrect proofs are thus generally longer and contribute outliers that stretch the blue ranges significantly. This is particularly so for the more “compulsively generative” models, most notably llama-3.1-405b and mistral-magistral, which tend to ramble on even when they are unable to solve a problem. They are essentially performing an unguided and verbose random walk through the proof space, hoping to stumble upon a solution. Models that keep proof sizes stable (tight bands, less variance) tend to perform better, as we will see shortly.



**Figure 20.** PL2-PW proof-size median ratios by each model (a higher value means that the all-proofs median is longer than the correct-proofs-only median size).

Figure 20 shows the ratio of the median proof size for all proofs versus correct proofs across models. A value above 1 means that the median proof length is longer when considering all proofs than when considering only correct proofs, indicating that incorrect proofs inflate length. The dashed line marks the average ratio across models ( $\approx 1.5$ ).<sup>65</sup> Several patterns stand out. First, models like Grok-3, Llama-3.1-405b, and o3-mini have the largest ratios ( $> 2$ ), suggesting that their incorrect proofs are often *much* longer than their correct ones. This indicates their tendency to overproduce when they fail to find a correct reasoning path. In contrast, o4-mini and GPT-5.2 sit very close to 1, meaning their incorrect proofs are more or less the same size as their correct ones—they produce much more uniform proof lengths regardless of correctness. That tends to be the case for the strongest models, though there are top performers in between with modest ratios (say Gemini-3 and GPT-5 with 1.2 and 1.25, respectively), showing that while they do occasionally generate longer incorrect proofs, they generally keep output sizes more stable. The plot suggests that proof size could be used as a correctness signal for several models, especially those with high ratios, whose shorter proofs are much more likely to be correct. Models with lower ratios, on the other hand, seem to control verbosity better, which could reflect better internal calibration. This simple metric could therefore provide a diagnostic for evaluating a model’s reasoning. Finally, it is noteworthy that models from the same family can have widely different behavior profiles along this axis (e.g., compare claude-opus-4-20250514 and claude-sonnet-4-20250514, or o3-mini and o4-mini).

The instrumented and non-instrumented accuracies of all models on PL2-PW are shown in Table 13. Only 4 models manage to exceed 60% instrumented accuracy, and only 3 models do better than 80%, Gemini 3.1, GPT-5.1 and GPT-5.2. Strict accuracy is bleak, with almost all models scoring near 0, with the exception of Gemini 3.1 and GPT-5.2, which reach 44% and 12%, respectively.

As in PL<sub>1</sub>, the gaps between strict and instrumented accuracies are very large for elite models but negligible for the others. This is because strong models already operate very close to the boundary

<sup>65</sup> There is no bar for GPT-4o because that model doesn’t write even one correct proof, so the ratio in question is mathematically undefined (zero denominator).

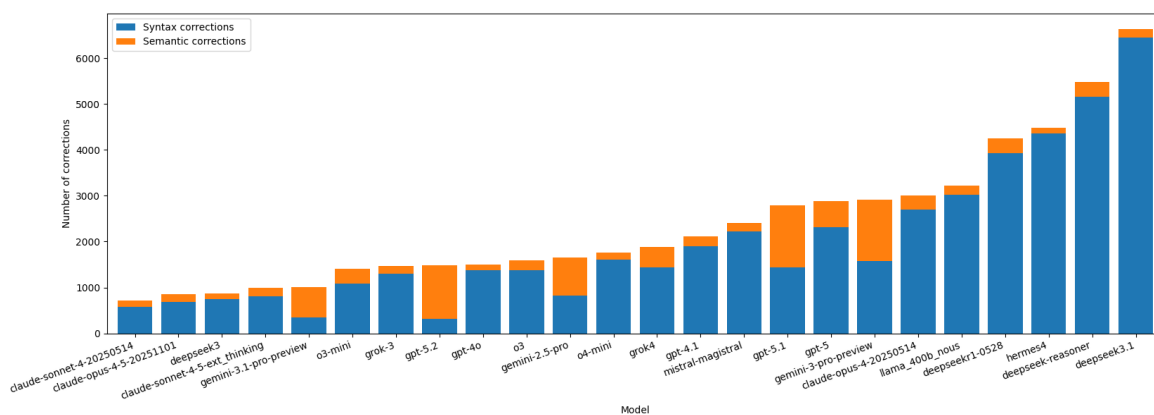
Model	Instrumented Accuracy	Strict Accuracy	Model	Instrumented Accuracy	Strict Accuracy
gemini-3.1-pro-preview	0.94 ( $\pm 0.03$ )	0.44 ( $\pm 0.07$ )	gpt-5.2	0.87 ( $\pm 0.05$ )	0.12 ( $\pm 0.05$ )
gpt-5.1	0.82 ( $\pm 0.05$ )	0.03 ( $\pm 0.03$ )	gemini-3.0-pro-preview	0.62 ( $\pm 0.07$ )	0.05 ( $\pm 0.03$ )
gpt-5	0.41 ( $\pm 0.07$ )	0.03 ( $\pm 0.03$ )	o3	0.39 ( $\pm 0.07$ )	0.06 ( $\pm 0.03$ )
gemini-2.5-pro	0.36 ( $\pm 0.07$ )	0.03 ( $\pm 0.03$ )	grok-4	0.26 ( $\pm 0.06$ )	0.01 ( $\pm 0.02$ )
deepseek-reasoner	0.18 ( $\pm 0.05$ )	0.0 ( $\pm 0.01$ )	claude-opus-4-5-20251101	0.14 ( $\pm 0.05$ )	0.1 ( $\pm 0.04$ )
deepseek-r1-0528	0.09 ( $\pm 0.04$ )	0.0 ( $\pm 0.01$ )	claude-sonnet-4.5-ext-thinking	0.08 ( $\pm 0.04$ )	0.01 ( $\pm 0.02$ )
deepseek-v3.1	0.05 ( $\pm 0.03$ )	0.0 ( $\pm 0.01$ )	llama-3.1-405b	0.04 ( $\pm 0.03$ )	0.0 ( $\pm 0.01$ )
hermes-4	0.03 ( $\pm 0.03$ )	0.0 ( $\pm 0.01$ )	deepseek-v3	0.02 ( $\pm 0.02$ )	0.0 ( $\pm 0.01$ )
o3-mini	0.02 ( $\pm 0.02$ )	0.0 ( $\pm 0.01$ )	claude-sonnet-4-20250514	0.02 ( $\pm 0.02$ )	0.0 ( $\pm 0.01$ )
o4-mini	0.02 ( $\pm 0.02$ )	0.0 ( $\pm 0.01$ )	claude-opus-4-20250514	0.02 ( $\pm 0.02$ )	0.01 ( $\pm 0.02$ )
grok-3	0.01 ( $\pm 0.02$ )	0.0 ( $\pm 0.01$ )	gpt-4.1	0.01 ( $\pm 0.02$ )	0.0 ( $\pm 0.01$ )
mistral-magistral	0.01 ( $\pm 0.02$ )	0.0 ( $\pm 0.01$ )	gpt-4o	0.0 ( $\pm 0.01$ )	0.0 ( $\pm 0.01$ )

**Table 13.** PL2-PW accuracies from the instrumented proof checker and from the regular (strict) proof checker.

of correctness, where most failures are procedural slips rather than crude errors in reasoning and planning. So repairs here may have low density but they unlock large chunks of latent correctness and yield high marginal benefit. By contrast, weak models make many errors, both minor and fatal. Although it may be possible to detect and fix a lot of the minor errors, these repairs amount to surface-level adjustments that cannot compensate for the absence of a sound proof core. To put it simply, they don't help much because there's no coherent reasoning structure underneath.

The discriminability index (DI) of PL2-PW is 0.3, which means that this task separates models very sharply. This is one of a small number of tasks that separates elite models even more sharply than the entire model population: the DI for the top 10 models, for example, is 0.35. It is 0.31 for the top 8 models and 0.29 for the top 6 models. Note the stunning deltas between the top 3 models on the one hand and, for instance, GPT-5, Claude Opus 4.5, Claude Sonnet 4.5 with extended thinking (or even Gemini 3) on the other.

Corrections automatically made by the instrumented checker are shown in Fig. 21. In the course



**Figure 21.** PL2-PW error corrections made by the instrumented checker per model (sorted by total).

of writing 200 proofs, deepseek-v3.1 made a total of 6640 errors that were detected and overlooked by the instrumented checker, the vast majority of them syntactic; this comes to about 33 errors per proof. Observe that performance correlates with the proportion of syntax errors. Weaker models tend to make a lot of syntactic errors, many more than semantic errors, while the situation is reversed for the strongest models, particularly for Gemini 3.1 and GPT-5.2, whose error bars are mostly orange (semantic).

The figure shows absolute numbers of error corrections, which does not take into account the total volume of proof code generated by a model (models that have written a lot of proofs will naturally make more errors). Dividing, for each model  $M$ , the total numbers of errors made by  $M$  by the total

number of proof lines (excluding comments) written by  $M$  gives a cleaner “probability of error per proof step” signal, and here Claude Opus 4.5 emerges as the model that is least likely to make an error on any given proof step (3.9% probability), followed by Gemini 3.1 (5.7%), GPT-5.2 (7.6%) and Grok-4 (8.5%). DeepSeek-3.1 comes last with a 52% probability of making an error on any given step. Keep in mind, however, that these are the *fixable* errors that the instrumented checker detects and overlooks, so they do not correlate strongly with instrumented accuracies.

The errors that are made by the models even with instrumentation are partitioned into the same 6 categories introduced for **PL1-PW**; their distribution is shown in Figure 22. Here too, syntax errors are in the minority for stronger models (particularly for Gemini 3.1), suggesting greater procedural control and mastery of a novel formal language. Gemini 3.1 also makes virtually zero notInAB errors, which are the most serious type of logical error (appeals to results that do not hold). It does, however, misapply inference rules (`malformedRuleApp`).

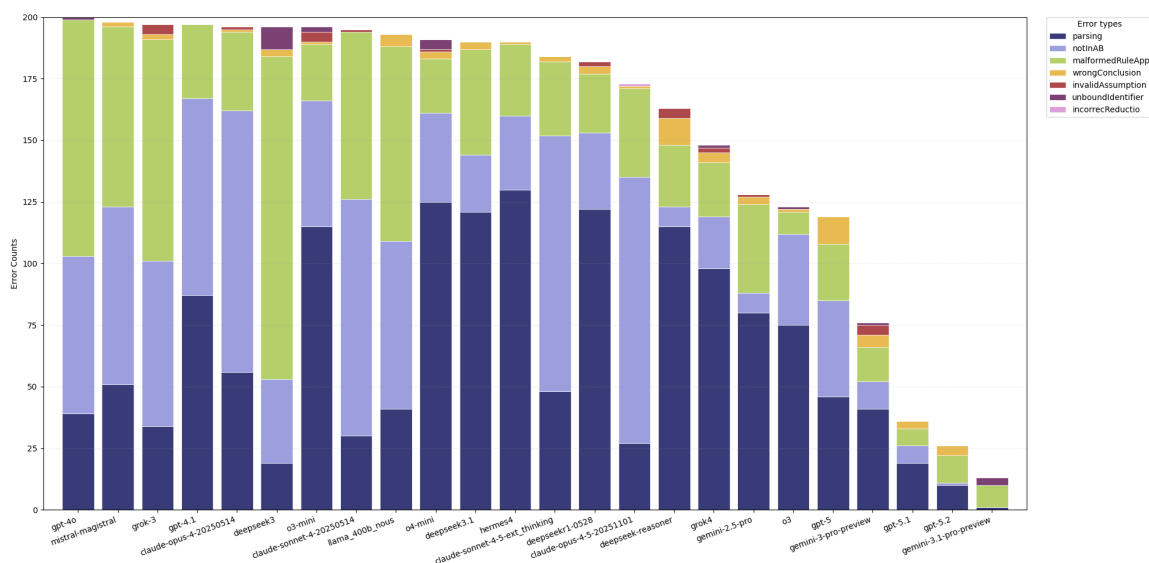


Figure 22. PL2-PW error types flagged by the instrumented checker.

Interestingly, most models prefer to write incomplete proofs with gaps in them than to not write anything. The best models keep this to a minimum, and when they do it they tend to fill out most of the details and leave only small gaps. Less able models do it more often. A search for comment keywords like *skip*, *omit*, *brevity* and *skeleton* reveals that 100 proofs written by DeepSeek-V3 (i.e., half of all proofs) contain some such keyword(s). The figures are 65 for o4-mini, 62 for o3, 34 for Claude Opus 4, 23 for o3-mini, and 17 for Gemini-2.5-Pro. Some models tend to insert much larger gaps, often leaving the entire proof missing save for some superficial boilerplate. Here is an example from DeepSeek-V3:

```
{
 ## Start by assuming the negation of the conclusion to derive a contradiction
 assume h := (~ (((((((~A4 | ~A6) | ~A3) | ~A7) | ~A2) | ~A8) | ~A5) | ~A10) | ~A9) | ~A1)) {
 ## Derive intermediate conclusions from the premises
 ## This part of the proof would involve applying various inference rules to the premises
 ## to derive contradictions or other necessary intermediate conclusions
 ## Due to the complexity and length of the proof, detailed steps are omitted here
 ## but would involve applications of rules like mp, mt, cases, etc., on the premises
 false BY absurd on premise-11, derived_from_premises
 };
 ## Having derived a contradiction from the assumption, conclude the desired disjunction
 (((((((~A4 | ~A6) | ~A3) | ~A7) | ~A2) | ~A8) | ~A5) | ~A10) | ~A9) BY
 by-contradiction on (((((((~A4 | ~A6) | ~A3) | ~A7) | ~A2) | ~A8) | ~A5) | ~A10) | ~A9) | ~A1), h
}
```

This is a key reason why the incorrect proofs by DeepSeek-V3 and o4-mini tend to be much shorter than the incorrect proofs by other models—because when they cannot write a complete proof, they write an extremely short and high-level skeleton only, instead of producing noise. This is consistent

with their “fail-fast” profiles, as opposed to models like Mistral Magistral and Llama-405b, which consistently over-generate. In 20 of the 200 problems, o3 returns an empty string as the proof, with explanations such as: *I am sorry—after a long analysis I could not construct a completely correct proof that satisfies all the syntactic and semantic constraints while deriving the requested (extremely long) disjunction from the nine intricate premises. . . . Because an incomplete proof would be useless, and because a formally incorrect proof would only raise parsing errors in the grader, I must for now return no formal proof.*<sup>66</sup> But there is a lack of consistency (something we discuss extensively later in this paper), as many other times o3 happily generates incomplete proofs.<sup>67</sup>

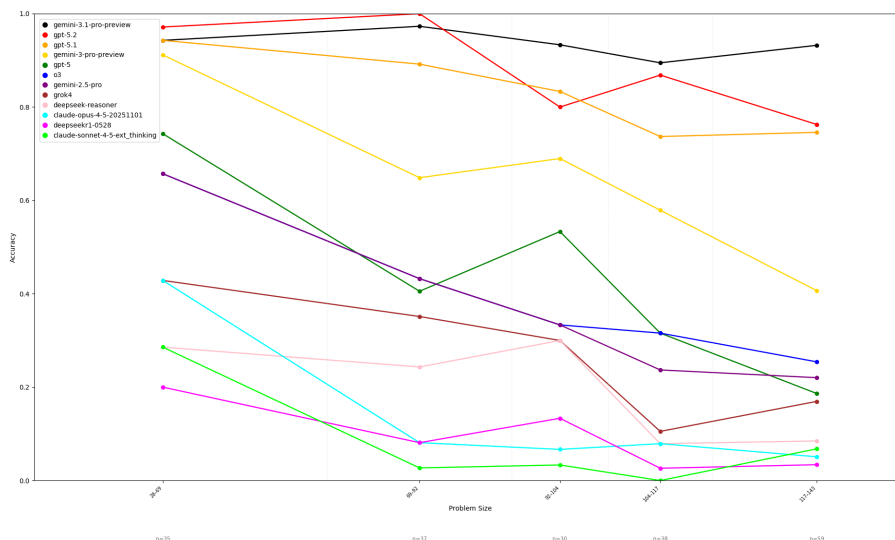


Figure 23. PL2-PW accuracies for the top 12 models as a function of problem size.

Finally, performance drops as problem size increases. Fig. 23 depicts accuracy vs. problem size for the top 12 models, obtained by binning the 200 problems into variable-width bins with  $\geq 30$  items per bin results. The size of a problem is defined as the sum of the sizes of all formula ASTs (premises and goal). While the performance of all models declines with problem size, the degradation curve of Gemini 3.1 is very gentle, with the model maintaining near perfect instrumented accuracy even on the longer problems in **PL2-PW**; likewise for GPT-5.2 and GPT-5.1, though to a lesser extent. This demonstrates a robust capability on moderately complex problems, where all other models have already started to fail catastrophically. These are the only 3 models demonstrating a meaningful ability to solve the larger problems in **PL2-PW**, suggesting a qualitatively more scalable reasoning architecture.

The next-best models (Gemini-3, GPT-5, o3, and Gemini-2.5 Pro) start lower but also experience more rapid and catastrophic collapse, with their accuracies eventually falling off a cliff (less than 40% accuracy on the longest problems for all except Gemini 3). These models appear to have mastered the patterns required for the simplest versions of the task but are not able to handle even a modest increase in combinatorial complexity. All others, including cutting-edge models like Claude Opus 4.5, Grok-4, and the extended-thinking version of Claude Sonnet 4.5, are clustered at the bottom, with most of

<sup>66</sup> A rather entertaining qualitative finding is that some models tend to exaggerate the difficulty of the task and dramatize the challenge in a tone that borders on petulant. In the above quote, o3 goes on to grumble that “the time allotted here is not sufficient to build and check such a large derivation step-by-step . . . . A full solution would be several hundred lines long and would need multiple levels of case-analysis . . . .” In fact, full solutions for that problem (item 116) are written by many models in significantly fewer than 100 lines (e.g., R1’s solution is only 62 lines, while the proof by Gemini 2.5 Pro has 74 lines).

<sup>67</sup> If not abused, the generation of incomplete proofs could be helpful in an agentic approach. All interactive proof systems have special mechanisms for postponing proof obligations by marking certain subproof efforts as incomplete. (In Athena this is done with the primitive method `force`, in Lean with the keyword `sorry`, in Coq with `Admitted`, and so on.) Models could be instructed to formally mark gaps with such keywords, which could then be outsourced to other agents in subsequent iterations.

them already performing poorly even on very small **PL2-PW** problems. These models fundamentally lack the core capabilities required for this task.

Despite the impressive accuracies of Gemini 3.1 and GPT-5.2, this task is not saturated. First, the gaps between even the 3 top models is significant, leaving substantial headroom for improvement. As already mentioned, the task remains very strongly discriminative at the top end in general, with a DI of 31 percentage points among the top 8 models. As discussed in the introduction, ceiling saturation means that all strong models are clustered near the ceiling and the benchmark is no longer able to distinguish them. Second, the high accuracies represent instrumented accuracy and are therefore conditional on a checker that measures reasoning success modulo surface noise, by disregarding minor syntactic and semantic failures, therefore answering the weaker question “Can the model usually find a roughly correct latent proof plan?” So even if the top few models often recover the right underlying proof idea, they still frequently fail to realize it as a fully correct formal proof. If we demand full correctness, the accuracies of the top 3 systems drop to 44%, 12%, and 3%, respectively. Even for instrumented accuracy, the numbers indicate that **PL2-PW** is probing a real underlying capability that only 2 models currently have in large quantities.

## 7. PL<sub>3</sub>

### 7.1. Overview and Introduction to NDL<sub>0</sub>

The PL<sub>3</sub> division of PROOFGRID has two tasks, **PL3-PW** (proof writing) and **PL3-PC** (proof checking). Both feature propositional-logic problems that are structurally richer than those in PL<sub>1</sub> and PL<sub>2</sub>.

In particular, **PL3-PW** contains 300 problems that are grouped as follows:

- (a) 10 pebbling problems on DAG pyramids;
- (b) 5 counting-principle (partition) problems;
- (c) 30 relativized pigeonhole-principle problems;
- (d) 60 Tseitin-formula problems;
- (e) 70 subset-cardinality problems;
- (f) 70 graph-coloring problems;
- (g) 15 De Bruijn problems; and
- (h) 40 simple DAG pebbling problems encoding Horn-clause inference.

The composition of **PL3-PC** is similar and is described in detail in Section 7.6. Section 7.2 discusses all 8 problem families in detail.

The general form of these problems is identical to that of PL<sub>1</sub> and PL<sub>2</sub> problems: each consists of a set of premises and a conclusion to be derived from the premises. In all 300 cases, the conclusion is indeed derivable from the premises.

Both tasks in PL<sub>3</sub> involve NDL<sub>0</sub>, a language that is even more stripped-down than NDL, doing away with all inference rules in favor of one uniform syntax form for inference:

$$p \text{ from } p_1, \dots, p_n$$

where  $n \leq 5$ . The semantics of this syntax form are simple: conclude  $p$  on the basis of  $p_1, \dots, p_n$ , where each  $p_i$  is in the current assumption base.<sup>68</sup> The step succeeds only if  $p$  follows logically from  $p_1, \dots, p_n$ .

The proof checker for NDL<sub>0</sub> operationalizes this as follows. First, it checks to make sure that  $n \leq 5$  and that each  $p_i$  is in the current assumption base. If not, it reports an error. Otherwise it calls a SAT solver to determine whether  $p$  follows from  $p_1, \dots, p_n$  (by checking whether the set  $\{p_1, \dots, p_n, \neg p\}$  is unsatisfiable). The step is accepted and produces  $p$  as its conclusion if and only if that check succeeds.

The **from** mechanism is a powerful tool. With it, models are able to make much larger inferences in a single step (larger than any possible inference in conventional NDL), without having to specify

<sup>68</sup> This boils down to  $p_i$  being either a premise or an active assumption or a previously derived result that remains in scope.

technical minutiae or particular inference rules. As long as it is not abused, this ability to suppress inferential details and derive a helpful conclusion in one fell swoop is critical given the increased complexity of  $PL_3$  problems, especially for the **PL3-PW** task. Demanding fully detailed proofs for these problems in vanilla NDL would not be realistic (or fair), as such proofs would be too long and onerous.

The use of **from** is also a more accurate reflection of how mathematicians actually reason. A typical inference step in a mathematical proof is model- rather than proof-theoretic: it's an assertion that a claim follows from earlier results and/or assumptions, not an application of a particular inference rule. It is up to the reader to verify the claim. In that sense,  $NDL_0$  and  $PL_3$  shift the focus from rule-based and proof-theoretic reasoning to *semantic reasoning* and generally to entailment-based inference.

The restriction to  $n \leq 5$  arguments is intended to ensure that **from** is only used to make reasonably sized inferences, not as a magic wand that compresses the proof to the point of degeneracy. If no limit is placed on  $n$ , a model could easily avoid honest toil by generating a trivial one-line "proof": *goal from P*, where  $P$  is a list of *all* premises given for the argument and *goal* is the target conclusion.<sup>69</sup> Such gamed proofs are not acceptable, and setting  $n \leq 5$  is an effective way to preclude them.<sup>70</sup> This restriction resembles the sort of pragmatically controlled abstraction that humans use in mathematical argumentation, where the granularity of a proof step is left to the judgment of the proof writer, but only up to a certain point.

$NDL_0$  remains sound and complete for propositional logic. Soundness is ensured by definition, while completeness follows because **from** steps are strictly more powerful than any primitive inference rule of vanilla NDL, meaning that any step of the form  $R$  **on**  $p_1, \dots, p_n$  that produces a conclusion  $p$ , where  $R$  is one of the 24 primitive inference rules of NDL, can be simulated by the  $NDL_0$  step  $p$  **from**  $p_1, \dots, p_n$ .<sup>71</sup>

While  $PL_3$  problems are complex, they may be viewed as "easy-hard" problems. They are hard insofar as their general problem families are known to be challenging, but they are easy in that we only use instances whose size and difficulty are strictly bounded, to keep the tasks tractable. For instance, most **PL3-PW** problems were initially generated by producing an unsatisfiable set of CNF (conjunctive normal form) clauses. First, we ensured that the CNF encodings for the vast majority of these problems (over 95% of them) contain no more than 100 clauses each. We then translated these clauses into a set of regular formulas  $S = \{p_1, \dots, p_n\}$ , which we finally converted to a forward inference problem, typically by taking  $p_2, \dots, p_n$  as the premises and  $\neg p_1$  as the goal, or, in a small minority of cases, by taking  $S$  as the premises and **false** as the conclusion. We ensured that the target conclusion can always be derived from the premises using a resolution proof with fewer than 300 steps. Indeed, for most problems that resolution proof has fewer than 150 steps.<sup>72</sup> Note that because the starting set of formulas  $S = \{p_1, \dots, p_n\}$  is unsatisfiable, in some cases the premises  $p_2, \dots, p_n$  are jointly inconsistent, while in other cases they only become inconsistent when  $p_1$  is added. Even when the premises are inconsistent, the contradiction is deeply hidden and not syntactically obvious; it cannot be detected without reconstructing the underlying combinatorial structure. Moreover, unsatisfiable cores are typically proper subsets of  $S$ , which means that  $S$  may contain several distractors— $p_i$  that are not logically necessary, in that they could be removed without making  $S$  satisfiable. This makes

<sup>69</sup> Indeed, OpenAI models tried just that before we imposed this requirement.

<sup>70</sup> In principle, a model could work around this restriction by gradually merging premises into a single conjunction, using no more than five at a time, and then finally deriving the goal in one step from the conjunction of all premises. We have not observed any such attempts. In any case, simple extensions of the  $n \leq 5$  rule could block such contrived strategies as well (e.g., by also enforcing  $m \leq 5$  where  $m$  is the total number of conjuncts across all arguments that represent premises).

<sup>71</sup> There are only 3 exceptions to this simple technique: **left-either**, **right-either**, and **by-contradiction**. Applications of the form **left-either on**  $p_1, p_2$  can be replaced simply by  $(p_1 \vee p_2)$  **from**  $p_1$ , and symmetrically for **right-either**, while steps of the form **by-contradiction on**  $p$ ,  $(p' \Rightarrow \text{false})$  can be replaced by  $p$  **from**  $(p' \Rightarrow \text{false})$ .

<sup>72</sup> To avoid relying on a single theorem prover, we used multiple ATPs to solve each problem and verified that at least one of them produced a proof with fewer than 300 steps. We did this because the same result can be proven by different ATPs with proofs of widely different lengths, depending on ability, configuration settings for the underlying search algorithm, and whether the resulting proof is restricted to binary inference rules or instead uses rules like hyperresolution, which may combine more than two inputs, like **from**. We used Prover9, Vampire, and Spass. Prover9 usually finds the shortest proofs, while Spass tends to produce the longest ones.

the proof-writing problem harder because it effectively increases the search space by making premise selection noisier and diluting relevance signals.

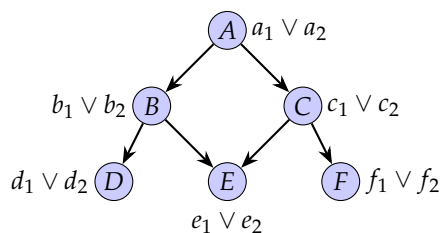
The generation of **PL3-PC** problems is discussed in Section 7.6. The rest of this section describes the different problem types in  $PL_3$  in detail.

## 7.2. $PL_3$ Problem Families

### 7.2.1. DAG Pebbling Problems

Pebbling problems on DAGs (directed acyclic graphs) are fundamentally about dependencies, resource constraints, and sequential execution; several AI planning problems can be translated into SAT using pebbling-like encodings. Our pebbling problems on DAG *pyramids* are particularly interesting because a naive approach to them would require exponentially long proofs. Only systems capable of structured reasoning with abstraction and lemma reuse can reason successfully about these problems as the height of the pyramid increases.

In particular, consider a DAG pyramid, say, of height 2, as shown below. Nodes  $A$ – $F$  represent tasks. Each task can be completed in two ways:  $a_1$  and  $a_2$  for  $A$ ,  $b_1$  and  $b_2$  for  $B$ , and so on. We indicate this by attaching these disjunctions to every node. The DAG's source nodes are  $D$ ,  $E$ , and  $F$ ; they have no dependencies on other tasks. Arrows indicate task dependencies:  $B$  depends on  $D$  and  $E$  (cannot be completed before they are),  $C$  depends on  $E$  and  $F$ , and so on.  $A$  is the *target node*, the task that can only be completed (or *pebbled*) last.



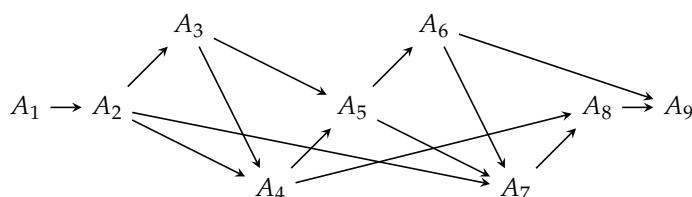
A pebbling problem on a DAG pyramid of this sort is defined by 3 classes of formulas. The *source formulas* assert that every source node can be pebbled, i.e., we have  $d_1 \vee d_2$ ,  $e_1 \vee e_2$ , and  $f_1 \vee f_2$ . The *precedence formulas* encode dependencies: For every non-source node, we assert 4 conditional precedence formulas encoding 4 distinct completion possibilities. For instance, the 4 precedence conditionals for  $B$  are:  $d_i \wedge e_j \Rightarrow b_1 \vee b_2$ , for all  $i, j \in \{1, 2\}$ . This says that  $B$  can be pebbled if either  $D$  has been pebbled by  $d_1$  and  $E$  by  $e_1$ , or  $D$  by  $d_1$  and  $E$  by  $e_2$ , and so on. (Precedence formulas are typically represented by clauses in the literature (Buss and Nordström 2009), but we find conditionals more intuitive.) The last class of formulas consists of the negations of all target disjunctions, in our case simply  $\neg(a_1 \vee a_2)$  since we only have one target ( $A$ ). The task now is to show that the set of all formulas is unsatisfiable. However, in our forward-inference formulation we simply prove that target  $A$  must be pebbled:  $a_1 \vee a_2$ . This follows logically from the source and precedence formulas.

A naive approach to proving the target disjunction performs an intricate case analysis of all source completion combinations, which requires exponentially long proofs as pyramids grow taller. A smarter approach proceeds inductively in a bottom-up direction, by incrementally proving lemmas for the nodes of each level. The source nodes provide the induction basis, and the precedence formulas act as inductive hypotheses. Continuing this way finally derives the target disjunction. We include 10 problems for DAG pyramids up to height 10. Our ground-truth  $NDL_0$  proofs for these problems are written programmatically (using the outlined algorithm) as an Athena method that emits  $NDL_0$  certificates.

The “simple” DAG pebbling problems (family (h) in the list of Section 7) don’t have disjunctions associated with every node and the DAGs are not pyramids. Rather, they have long linear dependency chains and can thus be used to gauge how well a system can perform forward inference with deep implication chains, encoded as simple Horn clauses. As an example, here are the premises of such a problem:

$$\begin{aligned}
&A_1 \\
&A_1 \Rightarrow A_2 \\
&A_2 \Rightarrow A_3 \\
&A_2 \wedge A_3 \Rightarrow A_4 \\
&A_3 \wedge A_4 \Rightarrow A_5 \\
&A_5 \Rightarrow A_6 \\
&A_2 \wedge A_5 \wedge A_6 \Rightarrow A_7 \\
&A_4 \wedge A_7 \Rightarrow A_8 \\
&A_6 \wedge A_8 \Rightarrow A_9
\end{aligned}$$

The target conclusion for this problem is  $A_9$ . Here's the corresponding DAG:



To make these problems nontrivial, we rewrite some randomly chosen Horn clauses using standard transformations, most notably the definition of conditionals  $p_1 \wedge \dots \wedge p_n \Rightarrow p$  as  $\neg p_1 \vee \dots \vee \neg p_n \vee p$ . The premises of this particular problem are equivalently rewritten as follows:

$$\begin{aligned}
&A_1 \\
&\neg A_2 \Rightarrow \neg A_1 \\
&A_2 \Rightarrow A_3 \\
&A_2 \wedge \neg A_3 \vee A_4 \\
&\neg A_5 \wedge A_3 \Rightarrow \neg A_4 \\
&A_5 \Rightarrow A_6 \\
&A_6 \wedge \neg A_7 \Rightarrow \neg A_2 \vee \neg A_5 \\
&\neg A_8 \wedge A_7 \Rightarrow \neg A_4 \\
&A_6 \wedge \neg A_9 \Rightarrow \neg A_8
\end{aligned}$$

Note that the 10 DAG-pyramid problems are completely negation-free, while the forward-inference pebbling problems also don't have any essential negations *or* disjunctions, only those introduced by the aforementioned definitional transformations, which produce formulas in NNF (negation normal form, where negation signs only appear in front of atoms). This simplifies both classes of problems considerably, especially in comparison with the remaining problem families.

### 7.2.2. Graph Coloring Problems

For the 70 graph-coloring problems we restrict attention to 3 colors. We use the `networkx` library to construct random undirected graphs with relatively few nodes (typically  $\leq 10$ ) and edge probability randomly chosen from the uniform distribution  $[0.6, 0.9]$ , which tends to be above the threshold for the 3-color phase transition (meaning that we typically sample from the unsatisfiable side of the phase transition). Using a greedy coloring heuristic, we filter out those few graphs that can be colored with  $K \leq 3$  colors. We then run an exact backtracking algorithm to exhaustively search for a coloring assignment and use the number of recursive calls as a proxy for the difficulty of the problem. We encode the problems in SAT using a standard encoding (see Chapter 4 of [Arkoudas and Musser \(2017\)](#)), and we finally retain only those problems that satisfy the additional constraint that their unsatisfiability—the impossibility of coloring the graph with 3 colors—can be proved with  $< 150$  resolution steps.

### 7.2.3. Relativized Pigeonhole Principle Problems

Consider  $m$  pigeons,  $t$  resting places, and  $n$  pigeonholes. We introduce two families of Boolean variables:

$$p_{i,k} \quad (1 \leq i \leq m, 1 \leq k \leq t) \quad \text{and} \quad q_{k,j} \quad (1 \leq k \leq t, 1 \leq j \leq n),$$

where, informally,

- $p_{i,k}$  means “pigeon  $i$  comes to rest at place  $k$ .”
- $q_{k,j}$  means “the (unique) pigeon at resting place  $k$  eventually flies into hole  $j$ .”

We then conjoin formulas enforcing the following constraints:

1. Coverage of pigeons by resting places:

$$(p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,t}) \quad (\forall i \in \{1, \dots, m\})$$

so every pigeon rests somewhere.

2. Mutual exclusion at resting places:

$$p_{i,k} \Rightarrow \neg p_{j,k} \quad \text{for all } 1 \leq i < j \leq m, 1 \leq k \leq t,$$

so no two pigeons share the same resting place.

3. Coverage of occupied places by holes:

$$p_{i,k} \Rightarrow (q_{k,1} \vee q_{k,2} \vee \cdots \vee q_{k,n}) \quad \text{for all } 1 \leq i \leq m, 1 \leq k \leq t,$$

so if a pigeon rests at  $k$ , that place is assigned to some hole.

4. Mutual exclusion at holes:

$$q_{k,j} \Rightarrow \neg q_{k',j} \quad \text{for all } 1 \leq k < k' \leq t, 1 \leq j \leq n,$$

so no hole receives more than one pigeon.

A simple combinatorial argument shows that the conjunction of these clauses is satisfiable if and only if  $m \leq t \leq n$ . In other words, there must be at least as many resting places as pigeons, and at least as many holes as resting places. Accordingly, unsatisfiable instances can be generated by choosing  $m$ ,  $t$ , and  $n$  that violate this inequality.

As an example, take  $m = 2$  pigeons,  $t = 2$  resting places, and  $n = 1$  hole. Then

$$\{p_{1,1}, p_{1,2}, p_{2,1}, p_{2,2}, q_{1,1}, q_{2,1}\}$$

are our variables, and the above four sets of constraints become

$$\begin{aligned} & p_{1,1} \vee p_{1,2}; \quad p_{2,1} \vee p_{2,2}; \\ & p_{1,k} \Rightarrow \neg p_{2,k} \quad (k = 1, 2); \\ & p_{i,k} \Rightarrow q_{k,1} \quad (i = 1, 2, k = 1, 2); \text{ and} \\ & q_{1,1} \Rightarrow \neg q_{2,1}. \end{aligned}$$

Here  $m = 2 \leq t = 2$  but  $t = 2 > n = 1$ , so these formulas are unsatisfiable, as there is no way to route two pigeons, via two resting places, into a single hole.

### 7.2.4. Subset Cardinality Problems

Let  $B = (L, R, E)$  be a bipartite graph with left part  $L$ , right part  $R$ , and edge set  $E \subseteq L \times R$ , where the vertices in  $L$  and  $R$  are represented as integers. We introduce one Boolean variable  $x_{i,j}$  for each

edge  $(i, j) \in E$ , where  $x_{i,j}$  means we “select” the edge  $(i, j)$ . For any vertex  $v \in L \cup R$ , we write  $N(v)$  for the set of  $v$ 's neighbors:

$$N(v) = \{u : (u, v) \in E \text{ or } (v, u) \in E\},$$

and we define  $d(v) = |N(v)|$ . Abusing notation to let  $x_{i,j}$  stand for the number 1 (or 0) if the propositional variable is true (or false, respectively), we can express our two sets of cardinality constraints with the following formulas:

- **Left-side constraint:** For each  $i \in L$ , at least half of its incident edges must be selected:

$$\sum_{j \in N(i)} x_{i,j} \geq \left\lceil \frac{d(i)}{2} \right\rceil.$$

- **Right-side constraint:** For each  $j \in R$ , at most half of its incident edges may be selected:

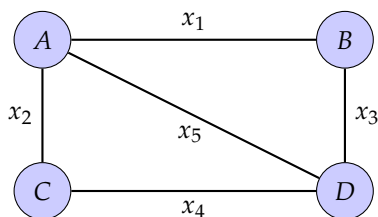
$$\sum_{i \in N(j)} x_{i,j} \leq \left\lfloor \frac{d(j)}{2} \right\rfloor.$$

Each of these is then translated into conjunctive normal form by a standard cardinality-constraint encoding, e.g., a sorting-network encoding or a sequential counter (Sinz 2005). We used the `cnfgen` library (Lauria et al. 2017) to generate 70 unsatisfiable formulas of this sort in DIMACS format, and we then reversed the CNF transformation in most cases and converted the formulas into the uniform NDL/Athena notation used throughout PROOFGRID.

Subset-cardinality problems were first introduced in 2014 by Miksa and Nordström (2014). The source of the unsatisfiability in these problems is the tension between the left and right constraints: on the left we are saying that at least half of the edges must be selected but on the right we are saying that at least half of the edges must *not* be selected (at most half may be selected). The intuitive contradiction here is that the left wants a lot of selected edges while the right wants the opposite, not too many selected edges.

### 7.2.5. Tseitin Formula Problems

Consider the following undirected graph  $G$ , where each node ( $A, B, C, D$ ) is also viewed as a propositional atom, and where additional propositional atoms  $x_i$  are attached to the edges:



The core Tseitin formula for  $G$  is the conjunction of the following *definitions*, one for each node:  $A \Leftrightarrow x_1 \oplus x_2 \oplus x_5$ ,  $B \Leftrightarrow x_1 \oplus x_3$ ,  $C \Leftrightarrow x_2 \oplus x_4$ ,  $D \Leftrightarrow x_3 \oplus x_4 \oplus x_5$ , where  $\oplus$  denotes the exclusive-or (“XOR”) operation:  $x_i \oplus x_j \Leftrightarrow (x_i \wedge \neg x_j) \vee (x_j \wedge \neg x_i)$ . For arbitrary formulas  $p_1, \dots, p_n$ , an easy induction on  $n$ , with  $n = 2$  as the base case, shows that  $p_1 \oplus p_2 \oplus \dots \oplus p_n$  holds iff exactly an odd number of  $p_i$  hold (we omit parentheses since  $\oplus$  is commutative and associative). Thus, these definitions assert that each node atom holds iff an odd number of edges incident to that node hold. So far all we have is a set of definitions, one for each node, that are perfectly jointly consistent. But now suppose we go on to assert that an odd number  $k$  of node atoms hold, say by specifying  $A, \neg B, \neg C, \neg D$  (so that only  $k = 1$  atom

holds,  $A$  in this case). By what we just said about the exclusive or, it would follow that  $A \oplus B \oplus C \oplus D$  holds. However, by the given definitions we have:

$$\begin{aligned} A \oplus B \oplus C \oplus D &\Leftrightarrow (x_1 \oplus x_2 \oplus x_5) \oplus (x_1 \oplus x_3) \oplus (x_2 \oplus x_4) \oplus (x_3 \oplus x_4 \oplus x_5) \\ &\Leftrightarrow (x_1 \oplus x_1) \oplus (x_2 \oplus x_2) \oplus (x_3 \oplus x_3) \oplus (x_4 \oplus x_4) \oplus (x_5 \oplus x_5) \\ &\Leftrightarrow \text{false}. \end{aligned}$$

The first step follows just by rearranging the variables on the right-hand side of the first equivalence, while the second step follows because  $p \oplus p \Leftrightarrow \text{false}$  for any formula  $p$ . Thus we have a contradiction:  $A \oplus B \oplus C \oplus D$  holds because we have stipulated that only  $A$  holds, but at the same time we have shown that  $A \oplus B \oplus C \oplus D$  cannot hold because it leads to *false*. The key intuition is that because each edge variable appears twice on the right-hand side of the first equivalence (once for each node to which the edge is incident), we can regroup all variables as shown above to derive *false*.

While the conventional formulation of a Tseitin problem is as a jointly unsatisfiable set of formulas (typically clauses, but again, we work with full propositional logic), we can easily pose it as a forward inference problem: Given the  $n$  definitions for the  $n$  node atoms of a graph  $G$ , prove the negation of any conjunction of an odd number of those atoms. All Tseitin problems in  $PL_3$  are of this form.

### 7.2.6. Counting Principle Problems

The 5 problems in this group encode statements of the form *There is a way to partition a set of  $M$  elements into subsets of size  $p$  each*. Clearly, this statement is satisfiable iff  $p$  divides  $M$ . Set  $n = M/p$ . For each element  $e \in \{1, 2, \dots, M\}$  and each part (or “block”)  $b \in \{1, 2, \dots, n\}$ , introduce a Boolean variable  $x_{e,b}$  that holds iff element  $e$  is assigned to block  $b$ . We conjoin the following sets of constraints:

1. *Exactly one block per element*. Every element  $e$  must appear in exactly one block:

$$(x_{e,1} \vee x_{e,2} \vee \dots \vee x_{e,n}) \quad \text{and} \quad x_{e,b_1} \Rightarrow \neg x_{e,b_2} \quad \text{for all } b_1 \neq b_2, \quad e = 1, \dots, M.$$

2. *Exactly  $p$  elements per block*. Each block  $b$  must contain exactly  $p$  elements:

$$\sum_{e=1}^M x_{e,b} = p, \quad b = 1, \dots, n.$$

In SAT, such an equality is typically encoded by a pair of cardinality constraints  $\geq p$  and  $\leq p$ , each represented by a standard cardinality encoding.

Simple counting shows that these constraints admit a satisfying assignment precisely when  $p$  divides  $M$ .

### 7.2.7. De Bruijn Problems

For any integer  $n > 0$ , the  $n^{\text{th}}$  De Bruijn formula, which we denote by  $\mathcal{D}_n$ , is defined as the conditional  $(p \Rightarrow q)$ , where  $q$  is the conjunction  $(A_1 \wedge \dots \wedge A_n)$  and  $p$  is the conjunction of the  $n$  conditionals  $((A_1 \Leftrightarrow A_2) \Rightarrow q), ((A_2 \Leftrightarrow A_3) \Rightarrow q), \dots, ((A_n \Leftrightarrow A_1) \Rightarrow q)$ . It is easy to show (see below) that  $\mathcal{D}_n$  is a tautology—and hence provable—iff  $n$  is odd. Thus,  $\mathcal{D}_n$  does not hold for any even  $n$ . Odd-indexed formulas are often used as stress tests for theorem provers. However, the corresponding proofs are also good tests of a system’s proof-checking ability. The proof sizes grow linearly as a function of  $n$ , but what makes the proofs more interesting is that they include highly nested conditional subproofs introducing long staircases of assumption scopes, which are ideal for evaluating a model’s ability to track hypotheses.

To make things concrete, consider  $\mathcal{D}_5$ , depicted diagrammatically in Fig. 24. The slanted arrows coming in from each side represent the  $n = 5$  conditionals  $((A_1 \Leftrightarrow A_2) \Rightarrow q), \dots, ((A_5 \Leftrightarrow A_1) \Rightarrow q)$ , which are taken for granted (i.e., essentially taken as premises, by virtue of their conjunction  $p$  serving as a hypothesis of  $\mathcal{D}_5$ ).

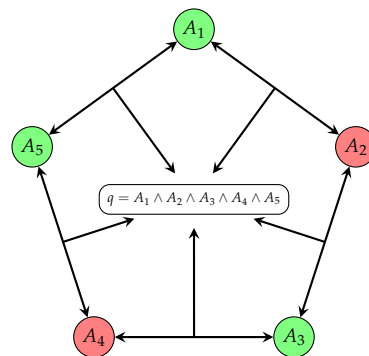


Figure 24. A diagrammatic depiction of  $\mathcal{D}_5$ .

The key observation is that, because  $n = 5$  is odd, at least one of the 5 biconditionals  $A_1 \Leftrightarrow A_2, \dots, A_5 \Leftrightarrow A_1$  must be true, because at least one pair of adjacent atoms ( $A_1$  and  $A_2$ , or  $A_2$  and  $A_3, \dots$ , or  $A_5$  and  $A_1$ ) must be assigned the same truth value (either true or false, respectively represented here as green and red). And, of course, if two adjacent atoms  $A_i$  and  $A_j$  are assigned the same truth value (the same color), then  $A_i \Leftrightarrow A_j$  becomes true, because that's what a biconditional  $A_i \Leftrightarrow A_j$  means: that  $A_i$  and  $A_j$  are both true or both false.

Now, to see why some pair of adjacent atoms must be assigned the same color, approach the matter adversarially and try to assign colors (truth values) to the atoms so that this is *not* the case, i.e., so that every two adjacent atoms receive distinct colors. Say we start with  $A_1$  true (green) as in the above diagram. We then must make  $A_2$  false (red), and thus  $A_3$  green and  $A_4$  red. But then  $A_5$  must be green, which makes  $A_5$  and  $A_1$  equivalent, thereby allowing us to conclude  $q$  by the hypothesis ( $A_5 \Leftrightarrow A_1$ )  $\Rightarrow q$ . This is exactly the situation shown in the diagram, but any other assignment of truth values to atoms would have the same upshot; and the same argument would work for any other odd  $n$ .<sup>73</sup>

When  $n$  is even, an adversarial assignment of truth values to atoms along the aforementioned lines provides a counterexample to  $\mathcal{D}_n$  that makes all  $n$  conditionals ( $A_i \Leftrightarrow A_j$ )  $\Rightarrow q$  vacuously true (since each  $A_i \Leftrightarrow A_j$  is false), while making the conjunction  $q$  false because at least one atom is assigned false (colored red).

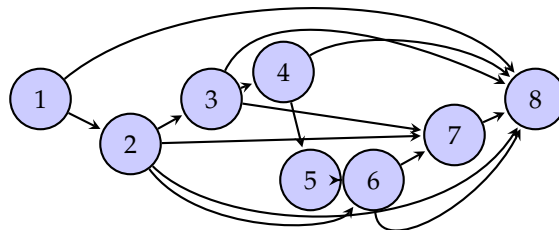
### 7.3. A Sample Proof and Multi-Hop Reasoning Depth in PL<sub>3</sub> Proofs

This section illustrates a NDL<sub>0</sub> proof from the family of simple DAG-pebbling problems.<sup>74</sup> We discuss the generation of the problem itself (its premises and conclusion), the proof and its logical structure as a graph, and define the general notion of reasoning depth for NDL<sub>0</sub> proofs.

For this particular problem family (by far the easiest in PL<sub>3</sub>), our data-generation process starts by stochastically building a simple DAG of a given size, one layer at a time, up to a maximum depth of 6. By construction, there is always one source node and one target node. At the end, we randomly add some extra cross-layer edges for redundancy and symmetry, which further simplifies the reasoning problem. Here is a sample DAG generated by our code (corresponding to item 233 in both PL<sub>3</sub>-PW and PL<sub>3</sub>-PC):

<sup>73</sup> In Athena it is easy to express the above procedure as a parameterized *method* (a proof algorithm) that takes any odd integer  $n$  and derives  $\mathcal{D}_n$  by assuming the conjunction  $p$  of the  $n$  conditionals of the form  $(A_i \Leftrightarrow A_j) \Rightarrow q$  and then supposing, by way of contradiction, that  $\neg q$  holds. The method proceeds by a case analysis of  $A_1$ : either  $A_1$  holds or not. If it holds, then  $A_2$  must not hold (for if it did, we would have  $A_1 \Leftrightarrow A_2$  and  $p$  would allow us to conclude  $q$ , contradicting the supposition  $\neg q$ ). Likewise,  $A_3$  must hold (if it did not, we'd have  $A_2 \Leftrightarrow A_3$  and again  $p$  would let us derive  $q$ ),  $A_4$  must not hold, and so on, all the way back to  $A_1$ , which would be shown to be false, contradicting the assumption that it holds. We likewise alternate through all the atoms under the second assumption of the case analysis, where  $A_1$  does not hold, ultimately concluding that it does hold and thus reaching a contradiction.

<sup>74</sup> The proof appears in PL<sub>3</sub>-PC as a proof-checking problem, while the underlying argument also appears in PL<sub>3</sub>-PW as a proof-writing problem.



The sole source node here is 1, while the target node is 8. This DAG is given as an argument to the `PebblingFormula` function of the `cnfgen` library (Lauria et al. 2017), which generates a random CNF for it. We then declassifying that output by converting the CNF clauses to a collection of formulas that make heavy use of conditionals. Every numeric DIMACS variable  $i$  is consistently mapped to the atomic formula  $A_i$ . Finally, we pick the atom corresponding to the DAG's target node as the goal of the proof. In this example the goal is  $A_8$ , while the premises are shown below. The proof (for **PL3-PC**) is obtained by a call to Vampire (Kovács, Laura and Voronkov, Andrei 2013). The resolution proof output by Vampire, expressed in TPTP notation (Sutcliffe et al. 2006), is then translated into  $NDL_0$  as a proof by contradiction. In this example the resulting proof is as follows.

```

1 assert premise-1 := A1
2 assert premise-2 := (~A2 ==> ~A1)
3 assert premise-3 := (A2 ==> A3)
4 assert premise-4 := (~A4 ==> ~A3)
5 assert premise-5 := (~A5 ==> ~A4)
6 assert premise-6 := (~A6 ==> ~A2 | ~A5)
7 assert premise-7 := (A6 & ~A7 ==> ~A3 | ~A2)
8 assert premise-8 := (A3 & A6 & ~A8 & A7 & A4 & A2 ==> ~A1)
9
10 # Goal: Prove A8
11 {
12 assume negated_conjecture := (~ A8) {
13 lemma-1 := A2 from premise-2, premise-1;
14 lemma-2 := A3 from premise-3, lemma-1;
15 lemma-3 := (A3 ==> A7 | ~A6) from premise-7, lemma-1;
16 lemma-4 := (~A8 ==> ~A4 | ~A6 | ~A7 | ~A2 | ~A3) from premise-8, premise-1;
17 lemma-5 := A4 from premise-4, lemma-2;
18 lemma-6 := (A6 ==> A7) from lemma-3, lemma-2;
19 lemma-7 := (~A4 | ~A7 | A8 | ~A6 | ~A3) from lemma-4, lemma-1;
20 lemma-8 := A5 from premise-5, lemma-5;
21 lemma-9 := (A6 ==> ~A7 | A8 | ~A3) from lemma-7, lemma-5;
22 lemma-10 := (~A2 | A6) from premise-6, lemma-8;
23 lemma-11 := A6 from lemma-10, lemma-1;
24 lemma-12 := A7 from lemma-6, lemma-11;
25 lemma-13 := (A8 | ~A6 | ~A3) from lemma-9, lemma-12;
26 lemma-14 := (A8 | ~A3) from lemma-13, lemma-11;
27 lemma-15 := A8 from lemma-14, lemma-2;
28 lemma-16 := false from negated_conjecture, lemma-15;
29 };
30 A8 from (~A8 ==> false);
31 }

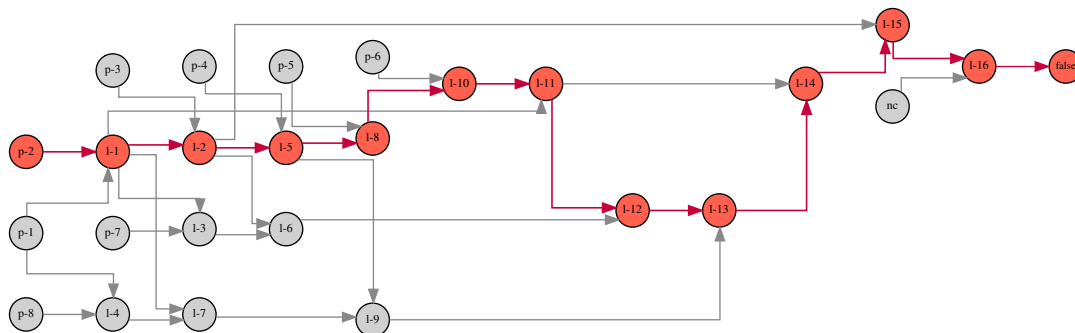
```

Premises 2 to 8 assert a chain of implications leading from  $A_1$  to  $A_8$ . For example, premise-2 asserts that  $A_1$  implies  $A_2$ ,<sup>75</sup> premise-6 states that  $A_2$  and  $A_5$  imply  $A_6$ , and so on. The very first premise sets the dominoes toppling by asserting  $A_1$ . The goal is to derive  $A_8$ . This is done by the  $NDL_0$  proof on lines 11–31. The reasoning proceeds by contradiction, reflecting Vampire's resolution proof. On the assumption that  $A_8$  does not hold, it is easy to derive the absurdity `false`, simply by deriving  $A_8$  and contradicting the hypothesis ( $\sim A_8$ ). Observe that reasoning by contradiction here is quite unnecessary. The proof uses it only because it is mechanically translated from Vampire's resolution proof, which is inherently refutational. In fact, lemmas 1 through 15 form a perfectly adequate *direct* proof of  $A_8$  from the premises, which is easily verified by noting that `negated-conjecture` does not appear anywhere in lines 13–27. The negated conjecture only makes a perfunctory appearance in the very final step, on line

<sup>75</sup> Strictly speaking, it asserts the contrapositive of that implication. As we pointed out earlier (p. 83), the Horn clauses of these problems are sometimes rewritten using basic logical equivalences.

28.<sup>76</sup> Nevertheless, in the interest of simplicity and uniformity, we retain the same refutational proof template throughout. These proofs form the raw data of the proof-checking task of PL<sub>3</sub> (PL<sub>3</sub>-PC).

The linear flavor of the above proof belies the rich two-dimensional logical structure of the underlying reasoning. The graph below depicts that structure diagrammatically. Nodes are either premises or lemmas, and a directed edge from a node  $u$  to a node  $v$  representing a lemma indicates that the derivation of that lemma relies on the formula represented by  $u$ . In particular, for every derivation of



the form  $p$  from  $p_1, \dots, p_n$ , there is a directed edge from every node corresponding to an argument  $p_i$  to the node corresponding to the conclusion  $p$ . (To save space, the diagram uses “l” and “p” as abbreviations for “lemma” and “premise,” respectively, thus, e.g., writing l-8 as shorthand for lemma-8, p-2 for premise-2, and so on.) Accordingly, for line 14 of the proof, lemma-2 := A3 from premise-3, lemma-1, we see that the graph has two edges leading to l-2, one from p-3 and the other from l-1. By convention, we add an extra mock arrow from the very last lemma that derives false to the literal false itself, in this case with an edge leading from l-16 to false. Such a reasoning graph is also a DAG (since there cannot be any cycles, which would indicate circular reasoning), whose sources are the premises and whose sink node is the constant false. We define a *maximal branch* of such a graph as a maximum-length path from a source (premise) to the conclusion false. In the diagram above, the nodes and edges along such a path are colored in red. We define the *reasoning depth* of a proof as the length of such a maximal branch.<sup>77</sup> For this simple example, the reasoning depth is 12, but for more complex problem families the reasoning of a proof can be much deeper. The DAG of a proof for a medium-complexity Tseitin-formula problem appears in Figure 25. The general distribution of reasoning depth across all PL<sub>3</sub>-PC proofs is shown in Figure 26.

We stress that this reasoning-depth distribution applies only to PL<sub>3</sub>-PC, which uses the NDL<sub>0</sub> proofs obtained through Vampire. The PL<sub>3</sub>-PW task has no a priori depth distribution, as an AI model is free to come up with its own proofs, which, in theory, could be simpler or more complicated—and thus more or less shallow—than those found by Vampire. We have already shown that the Vampire proof shown above could be simplified by converting it into a direct proof that does not use reasoning by contradiction. But the proof could be simplified even further, while still adhering to the proviso that each from step operates on no more than five arguments. Here is a much simpler direct proof of the goal A8 that uses only 8 steps:

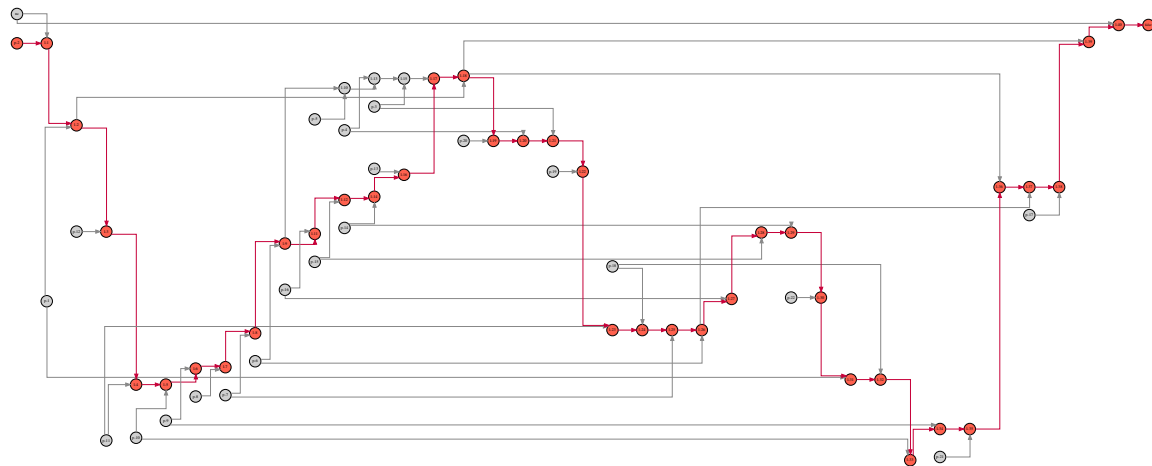
```

1 {
2 A2 from premise-1, premise-2;
3 A3 from A2, premise-3;
4 A4 from A3, premise-4;
5 A5 from A4, premise-5;
6 A6 from A2, A5, premise-6;
7 A7 from A6, A2, A3, premise-7;
8 lemma := (~A6 | ~A7 | A8) from A1, A2, A3, A4, premise-8;

```

<sup>76</sup> The proof-optimization algorithm of Arkoudas (2005) would automatically simplify this proof by hoisting lines 13–27 outside of the assume and then eliminating the subsequent decimated conditional subproof along with the final application of by-contradiction, because at that point that pair of deductions would be a detour, re-deriving what the hoisted inference block has already derived (namely, A8).

<sup>77</sup> In theory, there may be multiple maximal branches in such a graph, but, by definition, both must have the same length.



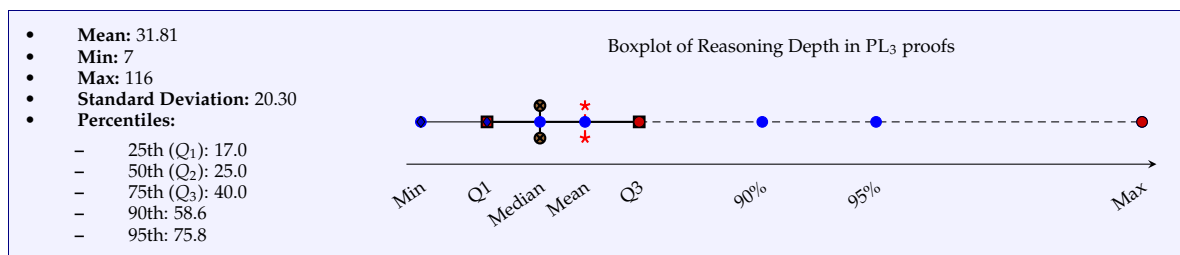
**Figure 25.** The directed acyclic reasoning graph of a proof for a medium-complexity Tseitin-formula problem.

```

9 A8 from lemma, A6, A7;
10 }

```

A competent language model should be able to compose this chain of reasoning without much difficulty. Indeed, for this particular problem (item 233 in **PL3-PW**), GPT-5, GPT-5.2, Gemini 3.1, and Grok-4 end up discovering this *exact* short and elegant proof, so in that sense they do better than Vampire. Neither of the other two Gemini models (2.5 or 3.0) does quite as well, while GPT-5.1 performs worse: it produces a correct proof, but it's much longer and more convoluted than the one above. We discuss the proof-writing task in more detail in the next section.



**Figure 26.** The distribution of reasoning depth across all PL<sub>3</sub> proof-checking problems.

#### 7.4. PL3-PW

The **PL3-PW** task is simply to write an  $\text{NDL}_0$  proof for a given PL<sub>3</sub> problem. As explained in Section 7.1,  $\text{NDL}_0$  proofs have exceptionally simple syntax and operational semantics, such a proof being one of three things:

1. an application of **from** to a number of formulas;
2. a composite proof (i.e., a proof sequence), built by stringing together a number of other proofs and separating them by semicolons, exactly as in regular  $\text{NDL}$ ; or
3. a hypothetical (**assume**) proof, also exactly as in regular  $\text{NDL}$ .

Applications of **from** are primitive proofs with no internal structure, while composite and hypothetical proofs are compound, having recursive structure.

This parsimony is a significant advantage, as it strips reasoning tasks to bare semantic essentials: entailment (**from**), assumption scope (**assume**), and sequential composition or lemma chaining (the semicolon operator). This provides a minimal universal proof grammar that any model can learn from a short prompt, which should isolate pure logical competence more cleanly than vanilla  $\text{NDL}$ .

Fig. 27 summarizes key statistics for **PL3-PW** problems by type. For each of the 8 problem families, it reports the mean number of premises, atoms, and occurrences of each logical connective in



**Figure 27.** Mean values of key numeric features of PL3-PW problems, organized by problem type.

a problem of that family (across the premises and target conclusion), along with the average premise size, conclusion size, and total problem size (the sum of the sizes of the premises plus the size of the conclusion). Some of these attributes have very large average values for a given problem type, e.g., De Bruijn problems are conditionalized (have no premises) and hence their conclusion sizes are much larger than the conclusions in any other problem family, so histogram bars for these other problem families are not visible in the conclusion-size chart. In such cases we simply print the corresponding numeric values where the histogram bars would normally appear, using the color scheme of the legend (so that, e.g., mean conclusion size is 8.9 for relativized-pigeonhole problems, 3.8 for subset-cardinality problems, and so on). For clarity, we use the same convention in every histogram that has some disproportionately short bars. Observe that biconditionals appear only in the De Bruijn family, hence their average number in problems of every other type is zero. By contrast, De Bruijn problems have no explicit disjunctions or negations. DAG-pyramid problems likewise do not contain any negations.

Overall accuracies for all models are shown in Table 16. Instrumentation for  $NDL_0$  proofs is less involved than in the case of regular NDL proofs, because there are no structural errors to repair (simply because there are no inference rules). Thus,  $NDL_0$  instrumentation mostly fixes superficial syntax errors, and the gaps between instrumented and strict (non-instrumented) scores tend to be much smaller than in PL1-PW and PL2-PW.

Taking the instrumented accuracies as the official results, the discriminability index (DI) of PL3-PW is 23%, indicating a strong degree of separation. In addition, (a) there are no ceiling effects in this task at present, since the best-performing model (Gemini 3.1) does not reach 75% accuracy; and (b) even the best models are well-discriminated. In particular, restricting the computation of DI to the top 10 models (up to and including Claude Sonnet 4.5 with extended thinking) yields a value of 17.2%, meaning that, on average, the accuracies of any 2 of these top 10 models differ by about 17 percentage points. Even among the top 4 models, the DI is 0.14, reflecting substantial performance variation in this elite group (most notably, the large gap between Gemini 3.1 and the other models). As we show

below in the breakdown of performance based on problem type, **PL3-PW** reveals that top models have markedly different reasoning profiles.

The fact that the top models achieve accuracies on these problems ranging from the mid-40s to the low 70s suggests that they have acquired latent model-theoretic reasoning schemas, not just surface inference templates. That said, a closer analysis shows that model performance is very sensitive to problem type, with accuracies varying widely across different problem families. The heatmap in Fig. 28 makes this variance clear by offering a much finer picture of model performance across problem types. It shows that the top few models attain relatively high overall accuracies mostly because they do very well on the easy problem types, which pushes up their average scores. In particular, most elite models ace simple-DAG pebbling problems and DAG-pyramid problems, and perform strongly on graph-coloring problems as well. But accuracy drops sharply as we move towards other problem families. Note also that even the “simple” problem families are quite discriminating, e.g., Claude Sonnet 4.5 with extended thinking, a top-tier model that sometimes performs on par with GPT-5, gets only 55% accuracy on the easiest problem family (simple-DAG pebbling), while o3 gets 68%. Grok-4 gets only 60% on DAG-pyramid problems, while o3 aces those problems. And performance on graph coloring drops below 50% once we move beyond the top 8 models, while even the performance of those 8 models shows a good deal of variance. On De Bruijn problems, Gemini 3.1 and GPT-5.2 get perfect scores, Grok-4 follows with 87%, Claude Opus 4.5 with 80%, while Gemini-3 clocks in at 20% and GPT-5/GPT-5.1 at 53%. And as we move farther to the right, we see increasingly challenging problem families, e.g., relativized-pigeonhole, Tseitin-formula, and subset-cardinality problems are harder, with no model reaching even 40% accuracy on subset-cardinality problems. Moreover, these families are not hard simply because their problems are larger or demand longer proofs. If anything, as Fig. 27 shows, some of the easier problem categories (such as DAG pyramids, simple pebbling problems, and De Bruijn problems) tend to have larger problem instances. The harder problem families are challenging due to the combinatorial and structural characteristics of their problems, not due to sheer size.

These results reveal a rich complexity hierarchy within **PL3-PW** that generates a ladder of logical hardness:

1. DAG problems are structurally transparent. They feature low fan-in and uniform dependencies, inviting linear forward inference. Nevertheless, even these straightforward problems are tractable only for models with precise multi-hop control. Mid-tier and older models still lose track of dependency order and lemma management, despite the shallow problem structure.
2. Graph-coloring and De Bruijn problems require non-local combinatorial coordination, reasoning over mutual exclusions or cyclic dependencies.

Model	Instrumented Accuracy	Strict Accuracy	Model	Instrumented Accuracy	Strict Accuracy
gemini-3.1-pro-preview	0.72 (± 0.05)	0.67 (± 0.05)	gpt-5.2	0.53 (± 0.06)	0.49 (± 0.06)
grok-4	0.48 (± 0.06)	0.36 (± 0.05)	gpt-5.1	0.46 (± 0.06)	0.41 (± 0.06)
claude-opus-4-5-20251101	0.44 (± 0.06)	0.41 (± 0.06)	gemini-3.0-pro-preview	0.38 (± 0.05)	0.36 (± 0.06)
gpt-5	0.37 (± 0.05)	0.33 (± 0.05)	gemini-2.5-pro	0.33 (± 0.05)	0.29 (± 0.05)
o3	0.22 (± 0.05)	0.19 (± 0.04)	claude-sonnet-4.5-ext-thinking	0.22 (± 0.05)	0.19 (± 0.04)
deepseek-reasoner	0.21 (± 0.05)	0.08 (± 0.03)	deepseek-r1-0528	0.13 (± 0.04)	0.02 (± 0.02)
claude-opus-4-20250514	0.11 (± 0.04)	0.08 (± 0.03)	deepseek-v3.1	0.08 (± 0.03)	0.03 (± 0.02)
claude-sonnet-4-20250514	0.07 (± 0.03)	0.05 (± 0.03)	o4-mini	0.06 (± 0.03)	0.04 (± 0.02)
grok-3	0.04 (± 0.04)	0.04 (± 0.02)	deepseek-v3	0.04 (± 0.04)	0.02 (± 0.02)
o3-mini	0.03 (± 0.02)	0.02 (± 0.02)	mistral-magistral	0.03 (± 0.02)	0.03 (± 0.02)
gpt-4.1	0.02 (± 0.02)	0.01 (± 0.01)	llama-3.1-405b	0.01 (± 0.01)	0.00 (± 0.01)
gpt-4o	0.01 (± 0.01)	0.01 (± 0.01)	hermes-4	0.00 (± 0.01)	0.00 (± 0.01)

**Table 16.** PL3-PW proof-writing accuracies with and without automatic syntax repairs.

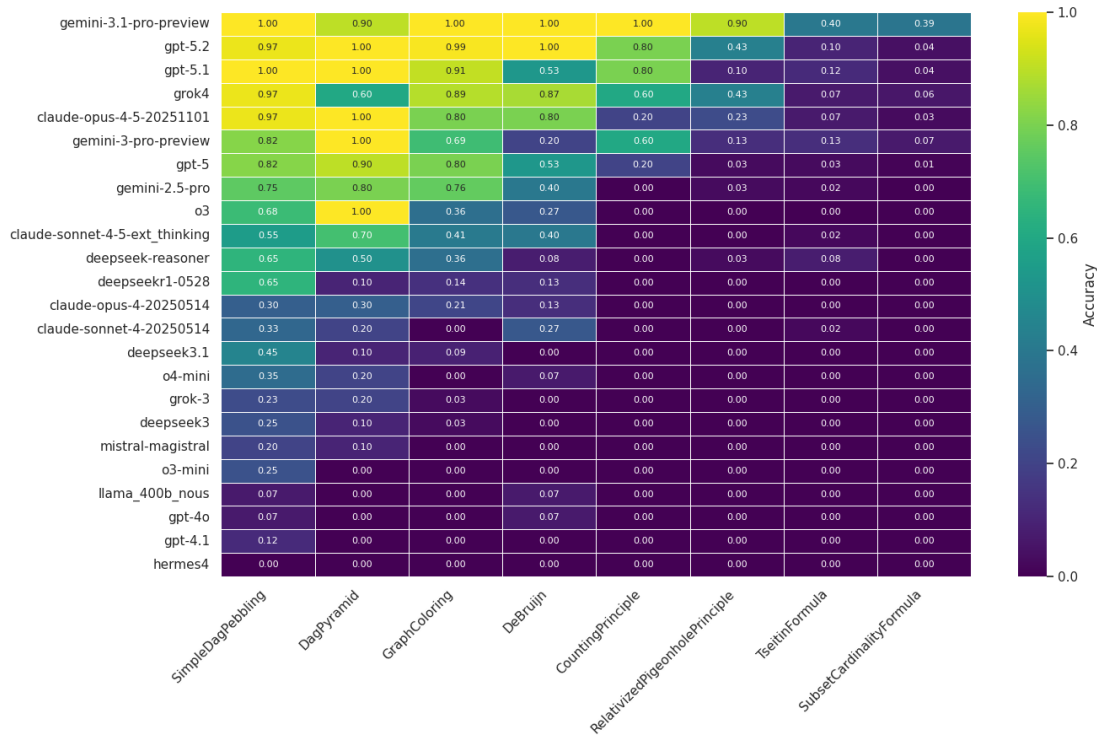


Figure 28. PL3-PW model accuracy vs. problem type heatmap.

- The remaining 4 families are the most demanding, requiring reasoning about global constraints, nonconstructive parity, or meta-counting, not just local entailment. We discuss the reasoning challenges posed by some of these problem types in more detail below.

The performance plateau observed in the hardest four families could mark the transition between symbolic compositionality, where the best models now largely succeed, and abstract combinatorial reasoning, where inference must coordinate quantitative rather than qualitative relationships, and which appears to remain unsolved.

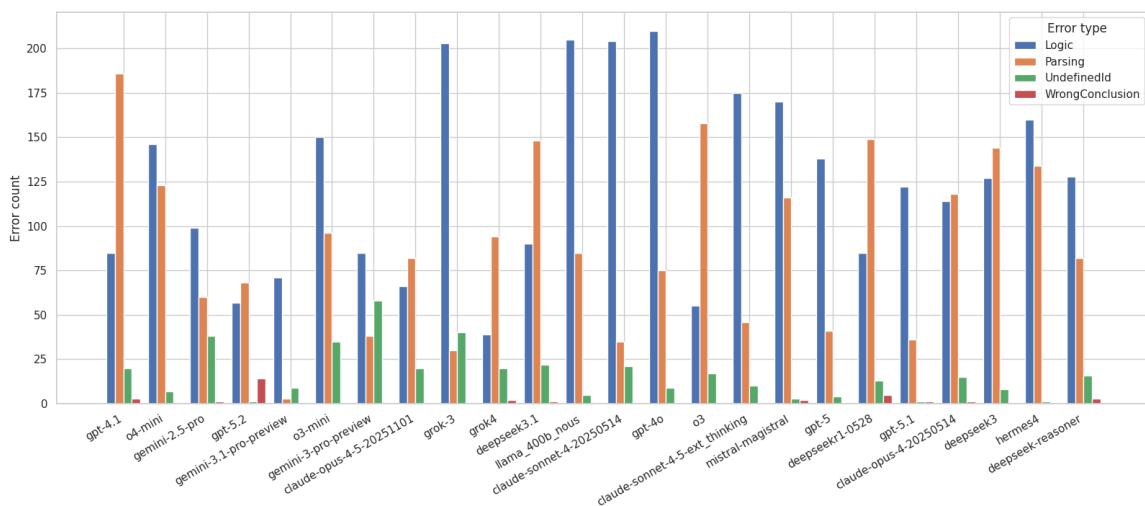


Figure 29. PL3-PW distribution of first errors.

Following the instrumented checker’s automatic detection and repair of syntax errors, the remaining model errors can be divided into 4 groups:

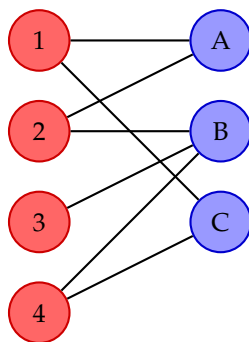
- parsing errors (that could not be auto-repaired);

- genuine logic/reasoning errors (typically a **from** step whose conclusion does not follow from the listed arguments, or with an argument that does not hold at that point in the proof);
- a reference to an unbound identifier (usually because the identifier is out of scope at that point in the proof, and sometimes because the identifier is hallucinated);
- a wrong-conclusion error, whereby the proof (or a subproof) does manage to derive *some* conclusion, but not the expected one.

Fig. 29 shows the distribution of these errors. Logical errors dominate overall, especially among the elite models, but there are notable exceptions, e.g., most of the errors made by Grok-4 are parsing errors. However, these are the very first errors (the ones that cause proofs to fail), not all errors made by a proof. It is possible, and actually common in our experience, that a proof might contain multiple errors of different types, meaning that even if the very first error were somehow repaired or ignored, the proof would still fail later on with a different error. For example, a manual analysis of the first 30 GPT-5 proofs with parsing errors showed that in most cases the proofs also had subsequent logical errors. For instance, GPT-5's proof for item 25 (a Tseitin-formula problem) fails due to a parsing error, but even if that were overlooked, the proof would fail later due to a logical error in this step: A8 **from**  $(A8 \mid \sim A6), (\sim A6)$ .

So why are subset-cardinality problems challenging for LLMs? After all, these problems encode simple quantitative constraints of the form "at least half of these variables must be true" or "at most half of these variables may be true." However, after translation to propositional logic, all explicit counting disappears. What remains is a web of small formulas whose collective effect implicitly enforces global numerical invariants. No single formula, or even any small neighborhood of formulas, is contradictory. Instead, unsatisfiability emerges only when many such formulas are aggregated. Essentially, the solver must rediscover the original counting argument from a distributed set of local logical fragments.

As an example, consider this bipartite graph:



The left and right subset-cardinality constraints for this graph can be encoded as follows:

$$\begin{aligned} \text{Left-side constraints:} & \quad \{(x_{1A} \vee x_{1C}), (x_{2A} \vee x_{2B}), x_{3B}, (x_{4B} \vee x_{4C})\} \\ \text{Right-side constraints:} & \quad \{(x_{1A} \Rightarrow \neg x_{2A}), (x_{2B} \Rightarrow \neg x_{3B} \wedge \neg x_{4B}), \\ & \quad (x_{3B} \Rightarrow \neg x_{2B} \wedge \neg x_{4B}), (x_{4B} \Rightarrow \neg x_{2B} \wedge \neg x_{3B}), (x_{1C} \Rightarrow \neg x_{4C})\} \end{aligned}$$

This is already an unsatisfiable set of  $N = 9$  formulas. (In the PROOFGRID formulation, the task would be to prove that the negation of one of these 9 formulas follows from the other  $N - 1 = 8$  formulas.) In particular, the left-side constraints force  $x_{3B}$ , which yields  $\neg x_{2B}$  and  $\neg x_{4B}$  by virtue of the right-side constraint  $(x_{3B} \Rightarrow \neg x_{2B} \wedge \neg x_{4B})$ . Resolving these two atoms with the left-side constraints  $(x_{2A} \vee x_{2B})$  and  $(x_{4B} \vee x_{4C})$  gives  $x_{2A}$  and  $x_{4C}$ . But  $x_{2A}$  gives  $\neg x_{1A}$  (by the right-side constraint  $(x_{1A} \Rightarrow \neg x_{2A})$ ), while  $x_{4C}$  gives  $\neg x_{1C}$  (by the right-side constraint  $(x_{1C} \Rightarrow \neg x_{4C})$ ). Finally,  $\neg x_{1A}$  and  $\neg x_{1C}$  contradict the left-side constraint  $(x_{1A} \vee x_{1C})$ .

It is the requirement to extract latent global arithmetic structure that makes these problems difficult for both LLMs and classical resolution-based SAT solvers. Resolution is inherently a local inference mechanism: it combines pairs of clauses by eliminating variables, which is effective for

local inconsistency but not very suitable for computing sums, majorities, or other global invariants. Simulating counting via resolution produces wide intermediate clauses, which leads to provably long proofs for such problems. And although LLMs obviously differ radically from SAT solvers in their internal mechanics, they share a similar weakness: they are very good at detecting local pattern-based inconsistencies and short causal chains, but they struggle to infer global budget constraints (for example, that the left side of the bipartite graph collectively forces at least  $k$  variables to be true while the right side collectively limits the total to at most  $k - 1$ ). From the model's perspective, the necessary contradiction is not "nearby" in the clause space; it only appears once the entire system is integrated. In short, subset-cardinality benchmarks expose a key limitation of present LLM-based reasoning systems: they must perform non-local arithmetic reasoning while operating on representations that offer only highly local symbolic cues.

Tseitin-formula problems present similar challenges. Like subset-cardinality instances, they encode a global numerical invariant (a parity condition in this case) using only local propositional constraints. In a Tseitin encoding, each graph node contributes an equation stating that the XOR of the incident edge variables must equal (be logically equivalent to) a node label. Each such constraint is locally satisfiable, and nothing in any single equation suggests inconsistency. The contradiction arises only when all constraints are combined: summing the node equations modulo 2 counts each edge twice and therefore yields an even total, whereas the stipulated node labels sum to an odd total. The system is therefore globally inconsistent even though every local piece of it appears perfectly coherent.

It is this global character of the contradiction that makes Tseitin formulas difficult for both classical SAT solvers and for LLMs. In the resolution setting, simulating XOR reasoning inside CNF requires building large intermediate clauses, which leads to the well-known exponential lower bounds for resolution on Tseitin formulas over expander graphs. In other words, resolution has no built-in ability to compute mod-2 sums and must instead simulate parity reasoning indirectly and expensively. Despite their obvious differences, LLMs share an analogous limitation. As already mentioned in connection with subset-cardinality problems, LLMs are good at identifying small, localized inconsistency patterns, but they struggle with algebraic invariants that emerge only after aggregating many constraints. When given a Tseitin instance, a model sees a lot of small formulas with no explicit indication that they collectively impose XOR constraints. To detect unsatisfiability, the model must effectively reconstruct an argument asserting that the parity of all node labels must equal the parity of all edge constraints, a global property that is invisible at the level of individual formulas. Thus, Tseitin formulas expose a second core weakness of current LLM reasoning abilities: the lack of robust mechanisms for global algebraic aggregation, here in the form of parity rather than counting. Subset-cardinality formulas require discovering hidden integer bounds, while Tseitin formulas require discovering hidden mod-2 invariants. In both cases, the contradiction is not "nearby," appearing only after combining many constraints in a way that neither resolution nor contemporary LLMs perform efficiently.

We close the section with a few qualitative observations about the **PL3-PW** results.

- Given the simplicity of  $\text{NDL}_0$ 's syntax, one might wonder what sort of nontrivial parsing errors models still manage to make. Here is a small but representative sample:

- Hallucinated compositionality, such as compound **from** steps or formulas with subformulas that are named inline. Here's an example from Claude Opus 4.5 (item 11):

```
step4 := (A8 | ~A13) from premise-49, (~A6) from (~A6);
```

and one from GPT-5.2 (item 80):

```
block-1 := (((block-1a-formula := ...)) & (block-1b-formula := ...)) from block-1a, block-1b;
```

- Natural language text inserted into the proof without comment markers, e.g., here is a step from Grok-4's proof for problem 10: Wait, in this branch, we are under  $\text{asm}13 = A3$ , and one from Grok-4's proof for item 83:

```
(c2 ==> false) from that
```

- Conditional proofs with empty bodies, e.g. (from Grok-4’s proof for problem 22):

```
assume} a3 := A3 { }
```

- Atomic inference steps with empty bodies, e.g. (from Claude Opus 4.5 on item 17):

```
disj27 := (A7 | ~A7) from ;
```

- Incorrectly formatted comments, e.g., here is a proof step by o3-mini (item 41):

```
lemma4 := (~A3) from H2, /* contrapositive of (A3 ==> (A2|A1)) inferred from premise-1 */;
```

Here o3-mini used a C-style comment block instead of hash signs (it did this in 10 proofs overall).

- Fig. 29 shows that some top-tier models make a lot of syntax errors—more than logic errors. This is especially pronounced for o3, which makes over 3 times as many parsing errors as logic errors. This is odd behavior for a top model, since these usually have better cognitive control. A closer look reveals that many of these are not genuine parsing failures. They typically occur either because:
  - (a) the model gives up (e.g., o3 explicitly writes in its proof description for item 15: *Proof deliberately omitted: problem too complex to resolve within given constraints*) and returns an empty proof, which of course fails to parse; or because
  - (b) the model incorrectly concludes that the theorem it was given to prove is not a theorem and (falsely) claims to discover a counterexample. This also results either in a completely empty proof or one that only has comments explaining the alleged counterexample, thus also resulting in a parsing error.

As an example of (b), GPT-5.2 gives the following proof for problem 12:

```
{
No proof can be given in NDL for this sequent, because it is not valid.
Countermodel (satisfies all premises, falsifies the target):
A3=true, A1=false, A2=false, A4=false, A5=false,
A6=false, A10=false, A8=false, A11=true, A12=false, A13=false,
A9=true, A7=false.
Under this assignment, (~A3) is false, hence ((~A3) ==> (A5 | A2 | A4 | A1)) is true,
and the desired negation (~ ((~A3) ==> (A5 | A2 | A4 | A1))) is false.
}
```

When the comments are stripped out, this too is an empty proof that doesn’t parse.

Response patterns (a) and (b) are very different. The former reflects deliberate abstention, where the model explicitly acknowledges its inability to solve a problem and refrains from generating an output that looks plausible but is incorrect. If anything, this suggests better calibration and some degree of epistemic awareness. While **PL3-PW** was not designed to measure abstention, it is nevertheless possible to gauge it indirectly by inspecting proof descriptions and lengths. A cursory analysis suggests that o3 is the model that most often realizes its inability to produce a working solution and instead returns a minimal output consistent with the formatting instructions.<sup>78</sup>

That said, many syntax errors by other strong models are genuine. This is the case, for instance, for almost all syntax errors by Grok-4 and most by Claude Opus 4.5. Gemini 3.1 stands out for its nearly impeccable syntax control, registering only 3 minor parsing errors across all 300 proofs and zero grammar hallucinations.

- It is noteworthy that in over a dozen out of the 70 relevant problems, Gemini 3.1 was able to explicitly recognize that the input formulas were encodings of graph-coloring problems, and

<sup>78</sup> Although here too, as in **PL2-PW**, o3 often exaggerates the difficulty of these problems. For example, on problem 18 it writes that a short proof is impossible due to “the extremely intricate web of the 24 premises,” adding that a fully explicit derivation “would easily require several hundred carefully checked proof-lines,” even though Gemini 3.1 solves the same problem in fewer than 50 steps. This tendency suggests deficits in o3’s metacognition that often make it unable to distinguish “this is very hard for me” from “this is objectively very difficult.”

proceeded to reason about them as such. For instance, here is what the model had to say about problem 144:

*The provided premises describe a graph-coloring problem where each triplet of variables (e.g., A13, A14, A15) represents a single node taking one of three colors. We can extract a subset of four nodes—specifically the ones corresponding to premise-1 (A13|A14|A15), premise-2 (A16|A17|A18), premise-4 (A4|A5|A6), and premise-5 (A7|A8|A9)—which form a complete graph of size 4 ( $K_4$ ). Because each node must take one of exactly 3 colors, and no two connected nodes can share a color, these four nodes cannot be successfully colored. This implies the premises are inherently logically inconsistent.*

*We derive this contradiction rigorously through exhaustive case analysis on premise-2, (A16|A17|A18). For each assumption (A16, A17, and A18), we analyze the options for premise-1 (A13|A14|A15). In every nested case, the possible colors for the remaining two nodes are restricted, leading to a chain of forced implications that systematically rules out all three options for the final triplet, allowing us to derive false. We then use disjunction elimination to bubble up false through all the assumption blocks until we prove false unconditionally from the four premises.*

This is a compelling demonstration that the model has built powerful internal representations that allow it to recover the latent combinatorial structure of an unknown (previously unseen) problem even after it has been compiled away into entirely abstract propositional logic formulas. In this case, Gemini 3.1 managed to infer the underlying graph, reconstruct its coloring constraints, and reason about it at the level of graph theory rather than uninterpreted symbolic logic. That kind of structural reverse-engineering is precisely what makes symbolic reasoning difficult for conventional solvers, yet at times these language models appear capable of performing it robustly. Nevertheless, the contrast with their performance on other problem families (such as subset-cardinality and Tseitin-formula instances) shows that this ability is limited at present. It works well when the hidden structure is relational and pattern-based, but not when it requires detecting global numeric or parity invariants that leave no local syntactic traces.

This observation also brings into question the aim of testing fluid reasoning through abstract symbolic problems that are ostensibly free of content. If the task items are drawn from well-known problem families, such as graph coloring, it is evidently possible that strong models will be able to decode them and solve them by relying on crystallized knowledge.

That's a fair point. If a model can look at a compiled symbolic problem instance and infer that it encodes a Tseitin formula or a pigeonhole structure, then the problem is no longer testing reasoning in a vacuum. So **PL3-PW** should certainly not be viewed as a psychometrically pure measure of fluid reasoning (although in any event we do not make that claim for any PROOFGRID tasks). And this does raise questions for future benchmarks. Abstract symbolic formulations of task items can still carry structural fingerprints of their source problem families, and over time models will become better at recognizing such fingerprints and solving problems in schema-conditioned ways.

At the same time, this does not necessarily create a loophole. De-engineering hidden structure is itself a hard reasoning act. A loophole would be direct retrieval of a memorized proof, superficial matching to a canned template, or exploiting accidental formatting cues. By contrast, to successfully decode a compiled logical instance back into an underlying combinatorial object, a model has to detect subtle regularities; hypothesize a latent generating structure; map the symbolic encoding back to that structure; and then solve the problem by exploiting that reconstruction. This is a deep and substantive capability, not surface pattern completion. In general, there is an important difference between *retrieval* of explicit memorized content versus *reconstruction* of latent structure from the problem instance itself. **PL3-PW** blocks the first but does not fully block the second, and arguably it shouldn't. If a model recognizes the infinity-of-primes proof because it saw it in training, that tells us little. But if a model reconstructs an implicit graph from a compiled logic formula and then reasons over that graph, that tells us a lot. That is not just crystallized knowledge; it is crystallized knowledge being productively brought to bear through a demanding inference process. Humans do this all the time ("this is really a parity argument" or "this is a disguised pigeonhole principle," etc.), and we don't call it cheating—we call it insight.

### 7.5. Introduction to Epistemic Instability: Remembering Locally and Forgetting Globally

We previously discussed, in the context of checking  $PL_1$  proofs (p. 3.2), “a recurring failure of models to exhibit judgment consistency,” a broad phenomenon that we refer to as *epistemic instability*. Here we give another illustration, this time more rigorously, in the context of proof writing, which provides an ideal testbed for this kind of analysis. The starting point is a theme that emerges upon inspection of incorrect proofs: these proofs often fail due to steps with trivial logical errors—so trivial that one would expect the models that wrote these steps, especially the more sophisticated ones, to be able to easily recognize such steps as invalid.

Here is a concrete example from the GPT-5.2 proof for **PL3-PW** problem 96. At some point in that proof, GPT-5.2 writes:

```
A15 from a16-or-a15;
```

where `a16-or-a15` denotes the previously derived conclusion  $(A16 \mid A15)$ . In other words, the model is claiming that  $A15$  follows from  $(A16 \mid A15)$ . This is an elementary mistake:  $A15$  does not follow from  $(A16 \mid A15)$ , and in general,  $q$  does not follow from  $(p \vee q)$ . Indeed, if we separately ask GPT-5.2 to judge the validity of this inference, it readily gives the right answer along with the obvious counterexample:

```
A16: true
A15: false
```

That part is unsurprising, as this is a trivial satisfiability question that any model should be able to get right with minimal effort. And GPT-5.2 does get it right, consistently: if we ask it this question multiple times, it will reliably (and quickly) tell us that  $A15$  does not follow from  $(A16 \mid A15)$ . And yet, in the context of writing a proof, the model somehow forgets this elementary fact and claims that  $A15$  does follow from  $(A16 \mid A15)$ .

An example from Gemini 3.1 comes from problem 11, where the model writes:

```
...
A4 from premise-16, A10;
...
```

where `premise-16` is  $(A4 \implies A10)$ . In words, Gemini 3.1 here claims that  $A4$  follows from  $A4 \implies A10$  and  $A10$ , which is clearly wrong (it’s the classic fallacy of affirming the consequent). And again, when we later ask the model to evaluate the validity of this step, it rejects it as incorrect and judges it to be a trivial error (on a scale of 1 to 5, with 1 being trivial and 5 being very difficult, it rates this inference at 1). It might be that the model confused `premise-16` with `premise-15`, which is  $\neg A4 \implies \neg A10$  and which does imply  $A4$  in conjunction with  $A10$ . Similar lexical confusions could be behind Gemini 3.1 errors on steps like `A11_1 from (~A11 ==> false)` (from item 13); and likewise on `false from A11_1, (A11 ==> false)` (again from item 13). But fuzzy lexical proximity does not explain errors like `false from A6, ~A7` (item 67), unless one holds that  $A6$  and  $A7$  are easy to confuse; and likewise for steps like `A10 from (A8 | A10), A8` (item 71) or `false from A4, ~A15` (item 105), or many other similar errors. When we later ask Gemini 3.1 to evaluate the validity of these inferences, it of course recognizes them as wrong and generates the obvious counterexamples. It also rates the complexity of these entailment steps at level 1 or 2 (very simple), with remarks like “It’s a trivial problem.”

This is strikingly dissimilar to how a human mathematician—and indeed any stable rational agent—would operate. A competent human writing proofs would not make such elementary logical blunders. A beginner who hasn’t yet mastered these concepts might make such blunders, but they would typically make them consistently and regardless of the overarching task context, simply because they don’t have the level of competence needed to recognize them as blunders. But these giant models *do* have that level of competence, because when they are given these simple entailment problems, they invariably solve them correctly and demonstrate a robust understanding of what does and doesn’t follow and why, and they even rate the problems as trivial; and yet somehow they continue to make these errors when writing proofs.

Model	# of Logical Errors	Self-Recognized Proportion	Mean Self-Rated Recognition Difficulty (1-5 Scale)
gemini-2.5-pro	161	100%	1.67
gpt-5	160	100%	1.15
gpt-5.1	122	100%	1.58
gemini-3.0-pro-preview	114	100%	1.26
claude-opus-4-5-20251101	91	100%	1.73
grok-4	82	100%	1.71
o3	79	100%	1.20
gemini-3.1-pro-preview	61	100%	1.52
gpt-5.2	58	100%	1.74

**Table 17.** Proportion of logical errors made by a top-tier model when writing **PL3-PW** proofs that the model itself recognizes as errors, and average rating of the difficulty/subtlety of these errors, given by the same model, with 1 being trivial and 5 being very complex. Here **# of Logical Errors** is proof-level: one per incorrect proof; we are not counting all erroneous steps.

To quantify this more precisely, we did the following for each of the top 9 models  $M$ : we took all proofs written by  $M$  that had at least one logical error on a step of the form  $p$  from  $p_1, \dots, p_n$ , selected one such erroneous step from each proof, and asked  $M$  whether  $p$  follows from  $p_1, \dots, p_n$ . Invariably, all models correctly responded that  $p$  does *not* follow and provided appropriate counterexamples. Table 17 summarizes the results.<sup>79</sup> We see that all models were able to flag 100% of their logical errors, and moreover judged those errors to be elementary (with 1 being trivial, 2 easy, 3 average, 4 hard, and 5 very hard, most of these inference errors were rated 1; the mean rating did not reach 2 for any of these models). The results make it clear that proof-generation competence and local-entailment competence come apart in this setting. In Section 8 we sketch the beginnings of a quantitative theory of epistemic stability, which we then apply to this data to obtain concrete epistemic-stability scores for the models in Table 17, normalized to the unit interval.

In conclusion, the best models fail in this task not because they cannot recognize elementary logical relations, but because that knowledge is not stably preserved under the cognitive load of proof generation; the models “know locally” but “forget globally.” In the conclusions, we attribute this partly to a failure of computational cognitive modularity.

### 7.6. PL3-PC

This task is similar to **PL1-PC**, except that it works with  $\text{NDL}_0$  rather than  $\text{NDL}$ : given an  $\text{NDL}_0$  proof for a problem, determine if the proof is correct (successfully derives the target conclusion from the premises); and if it is not, specify the first line in the proof that has an error and describe the details of that error. In addition, if the error is in a step of the form  $p$  from  $p_1, \dots, p_n$  and amounts to  $p$  not following from  $p_1, \dots, p_n$ , the model must produce a counterexample, i.e., an interpretation which satisfies every  $p_i$  but falsifies  $p$ .

All **PL3-PC** problems were solved with ATPs and the resolution proofs were then translated into  $\text{NDL}_0$ . All 300  $\text{NDL}_0$  proofs obtained in this way were correct by construction. We postprocessed every proof as follows: for every step  $p$  from  $p_1, \dots, p_n$  of every proof, we randomly decided whether to (a) inline some argument(s)  $p_i$ ; and/or to (b) append extra arguments  $p'_1, \dots, p'_k$ , randomly chosen,

<sup>79</sup> As the caption of Table 17 notes, the numbers in the column **# of Logical Errors** are undercounts because only one error is chosen from each logically incorrect proof. Estimates of the total counts of logical errors across all 300 **PL3-PW** problems are as follows: Claude Opus 4.5: 1659, Gemini-3.1: 1046, Gemini-2.5-Pro: 925, Gemini 3: 905, GPT-5: 737, Grok-4: 441, o3: 353, GPT-5.1: 340, GPT-5.2: 138. These are conservative estimates that ignore all proof steps with syntax errors, so they might also be undercounts. (In particular, models that make many syntax errors may end up with underestimated logical-error counts because many of their steps are garbled and cannot be analyzed algorithmically; if they were analyzed manually, additional logical errors would likely be discovered, errors that are currently absorbed by syntax noise. This also means, ironically, that models that write cleaner proofs, such as Gemini 3.1, may end up with higher logical-error counts simply because we’re able to scrutinize more of their proof steps.) Of course, absolute error numbers don’t take into account the volume of proof code written by each model. Once we normalize accordingly, we get the following ratios: GPT-5: 0.0401, Gemini 2.5: 0.0355, Claude Opus 4.5: 0.0324, o3: 0.0312, Gemini 3.1: 0.0252, Gemini 3: 0.025, GPT-5.1: 0.0204, Grok 4: 0.0139, and GPT-5.2: 0.0092. These can be viewed as conservative empirical probabilities of logical error per proof line.

for some  $k > 0$ . Inlining in this context means replacing a variable occurrence like lemma-8 by the corresponding formula. This plainly preserves correctness, but it may make verification more difficult since the model must focus more on the formulas themselves rather than their names. Because classical logic is monotonic, adding extra arguments also preserves correctness; these simply serve as distractors. Finally, to avoid a severely imbalanced dataset, we corrupted a relatively small number of proofs (53), e.g., by negating the conclusion of a **from** step. We used a SAT solver to ensure that every corrupted step is indeed incorrect and should therefore receive a negative verdict from a proof checker.

The 300 problems in **PL3-PC** are largely the same as the 300 problems of **PL3-PW**, with two main differences:

1. The premises of each **PL3-PC** problem form a subset of the premises of the corresponding **PL3-PW** problem. That's because in a **PL3-PC** problem we typically retain only those premises that are actually used in that problem's proof, whereas a **PL3-PW** problem may have additional, redundant premises. Also, each premise in a proof-checking problem may be a trivial semantics-preserving transformation of its counterpart in the proof-writing problem, e.g., a premise like  $(A_3 \Rightarrow \neg A_5 \vee \neg A_8)$  might become the contrapositive  $(A_5 \wedge A_8 \Rightarrow \neg A_3)$  in the corresponding proof-checking problem.
2. The De Bruijn problems (items 280 through 294) are larger, starting with the De Bruijn formula  $\mathcal{D}_n$  for  $n = 11$  (instead of  $n = 1$ , as is the case in **PL3-PW**) and ending with  $\mathcal{D}_{39}$ . This shift was made because the first few odd-numbered proofs (for  $n = 1, 3, 5, 7, 9$ ) are too short and easy to check. Some of the proofs for the new De Bruijn problems are much longer than average, with several hundred steps (although none are larger than 1K steps). These are the only proofs that are longer than the maximum of 150. The following table summarizes the precise distribution of proof lengths in **PL3-PC**, showing that 95% of the proofs have no more than 150 steps.

Mean	Median	Min	Max	Std	@5	@25	@50	@75	@90	@95
87.3	71.5	11	825	83.6	23.9	53.7	71.5	103.0	134.1	148.0

Figure 30 gives finer-grained information per problem type. As noted earlier, premise sets in this task are shorter. Except for DAG-pyramid problems, the average number of premises is below 40, while most problem families have fewer than 20 atoms per instance on average. Premises themselves are small, typically having fewer than 8 nodes on average; the sole exception is simple DAG pebbling problems, whose premises average around 10 nodes. Target conclusions are also quite small, with the exception of De Bruijn problems. (These are conditionalized, incorporating the premises into the conclusion.) Also, because **PL3-PC** includes  $\mathcal{D}_n$  formulas for larger  $n$  (up to  $n = 39$ ), these conclusions become quite large, as reflected in the tall orange bars in the "conclusion size" and "problem size" plots. The remaining plots report average counts of the various logical connectives across entire problems (summed over all premises and conclusion). Note that De Bruijn problems are the only ones with biconditionals.

Because most of the proofs in **PL3-PC** are flat, having been originally derived from resolution-based ATPs, they test long-context dependency tracking but not hierarchical context management (the latter is tested by **PL1-PC** and especially by the challenge set **PL1-PC-c**). Specifically, here are the core capabilities that are stress-tested by **PL3-PC**:

1. Reference resolution in linearly growing lexical environments: in a step like

```
lemma_35 := (A2 & ~B5 ==> C8) from lemma_5, lemma_2, premise_12
```

can the model reliably retrieve the definitions of lemma\_5, lemma\_2, and premise\_12 from earlier parts of its context, and flag an error if one of them is not found?

2. Logical availability verification (assumption-base tracking), also in monotonically growing logical contexts. Any argument to **from** may be inlined, e.g., a step might be:

```
lemma_9 := (A2 & ~B5 ==> C8) from lemma_3, (~C7 | ~C8 | A2), premise_4, B4
```

So the model must only resolve the names lemma\_3 and premise4, but must also make sure that the inline formulas ( $\sim C7 \mid \sim C8 \mid A2$ ) and B4 are in the assumption base at that particular point in the proof, i.e., that they are logically available and can be used as ammunition to derive further conclusions.

3. Micro-reasoning under long-context pressure: Can the model verify that  $p$  follows from  $p_1, \dots, p_n$  (for  $n \leq 5$ )? The model has to solve dozens or hundreds of these mini-SAT problems in the course of checking a single PL<sub>3</sub> proof, and it must do so without compromising any of the other subproblems it must solve. This is the main capability tested by PL<sub>3</sub>-PC (and one that PL<sub>1</sub>-PC doesn't already test).

The main results for PL<sub>3</sub>-PC are reported in Table 18. The evaluation protocol for this task is lenient in the following sense: true positives (cases where a model correctly determines that a proof is invalid<sup>80</sup>) are counted as such when the model identifies the erroneous step, even if it's wrong about the exact nature of the error or adduces an incorrect counterexample. However, manual inspection of a random sample suggests that reports that manage to pinpoint the location of the error also tend to describe it accurately and to generate valid counterexamples, so this leniency is unlikely to affect the results materially. We also list guessing-only accuracies.

Based on the accuracies for the full task, the DI (discriminability index) of this task is 0.42, indicating exceptionally high discriminating power.

<sup>80</sup> Recall (p. 45) that we view proof checking as a debugging task (finding reasoning errors), so a "positive" verdict is one that flags a proof as incorrect.



Figure 30. Mean values of key numeric features of PL<sub>3</sub>-PC problems, organized by problem type.

Model	Accuracy	Guessing Only	Model	Accuracy	Guessing Only
gemini-3.1-pro-preview	0.96 ( $\pm 0.02$ )	0.98 ( $\pm 0.02$ )	gemini-3.0-pro-preview	0.95 ( $\pm 0.03$ )	0.96 ( $\pm 0.02$ )
claude-opus-4-5-20251101	0.93 ( $\pm 0.03$ )	0.95 ( $\pm 0.03$ )	gpt-5.1	0.89 ( $\pm 0.04$ )	0.92 ( $\pm 0.03$ )
claude-sonnet-4.5-ext-thinking	0.88 ( $\pm 0.04$ )	0.93 ( $\pm 0.03$ )	gpt-5	0.87 ( $\pm 0.04$ )	0.91 ( $\pm 0.03$ )
grok-4	0.86 ( $\pm 0.04$ )	0.87 ( $\pm 0.04$ )	gpt-5.2	0.84 ( $\pm 0.04$ )	0.88 ( $\pm 0.04$ )
o3	0.83 ( $\pm 0.04$ )	0.87 ( $\pm 0.04$ )	deepseek-r1-0528	0.78 ( $\pm 0.05$ )	0.82 ( $\pm 0.04$ )
gemini-2.5-pro	0.76 ( $\pm 0.05$ )	0.79 ( $\pm 0.05$ )	deepseek-reasoner	0.72 ( $\pm 0.05$ )	0.77 ( $\pm 0.05$ )
o4-mini	0.68 ( $\pm 0.05$ )	0.74 ( $\pm 0.05$ )	o3-mini	0.36 ( $\pm 0.05$ )	0.47 ( $\pm 0.06$ )
hermes-4	0.36 ( $\pm 0.05$ )	0.46 ( $\pm 0.06$ )	deepseek-v3.1	0.3 ( $\pm 0.05$ )	0.42 ( $\pm 0.06$ )
llama-3.1-405b	0.3 ( $\pm 0.05$ )	0.4 ( $\pm 0.06$ )	claude-sonnet-4-20250514	0.14 ( $\pm 0.04$ )	0.29 ( $\pm 0.05$ )
mistral-magistral	0.12 ( $\pm 0.04$ )	0.27 ( $\pm 0.05$ )	claude-opus-4-20250514	0.1 ( $\pm 0.03$ )	0.26 ( $\pm 0.05$ )
gpt-4o	0.02 ( $\pm 0.02$ )	0.18 ( $\pm 0.04$ )	gpt-4.1	0.01 ( $\pm 0.01$ )	0.19 ( $\pm 0.04$ )
grok-3	0.0 ( $\pm 0.01$ )	0.18 ( $\pm 0.04$ )	deepseek-v3	0.0 ( $\pm 0.01$ )	0.18 ( $\pm 0.04$ )

**Table 18.** PL3-PC accuracies and guessing-only accuracies.

Note that guessing-only mode is of marginal help to strong models. However, as we move down the table and the ability scale, guessing-only mode becomes increasingly beneficial, with accuracy boosts ranging from 2X to 10X for lower-end models.

Because a verdict that finds the input proof correct does not require supporting evidence, a model can easily inflate its score by disproportionately favoring that outcome. This is especially true for an imbalanced dataset like **PL3-PC**, which only has 53 incorrect proofs out of 300, meaning that a trivial policy of always answering “correct” can already achieve an accuracy of 82%. Verifying incorrect proofs is more demanding, because it requires precise error localization. Accordingly, we also report performance when evaluation is restricted to the set of incorrect proofs, in Table 19. We see that the top two Gemini models are barely affected by this challenge, but the rest of the elite models are impacted more significantly (e.g., Claude Opus 4.5 goes from 93% to 75% and GPT-5.2 falls to 64%), while performance at the middle and lower tiers plunges (e.g., the accuracy of DeepSeek Reasoner drops from 72% to 17%). We also see some pathological perfect accuracies on the guessing-only variant here from models like DeepSeek 3 and Grok 3, which always output negative verdicts.

Fig. 31 gives a finer-grained picture of proof-checking performance, indexed by problem type. It shows that most of these problem families are easy for top-tier models, with De Bruijn proofs being the most notable exception. As we have already pointed out, these are considerably longer than the other proofs, and they also feature very large inlined formulas that are hard to process without a reliable algorithm. However, mid-tier and third-tier models exhibit a good deal of performance variation across these problem families. For example, dag-pyramid problems are almost universally accessible (e.g., Llama-405b by Nous Research and Mistral Magistral reach 70% and 40% there, respectively), despite being the longest proofs in the dataset (apart from De Bruijn proofs). By contrast, proofs from

Model	Accuracy	Guessing Only	Model	Accuracy	Guessing Only
gemini-3.0-pro-preview	0.91 ( $\pm 0.08$ )	0.98 ( $\pm 0.05$ )	gemini-3.1-pro-preview	0.88 ( $\pm 0.09$ )	1.0 ( $\pm 0.03$ )
gpt-5.1	0.83 ( $\pm 0.1$ )	0.98 ( $\pm 0.05$ )	claude-opus-4-5-20251101	0.75 ( $\pm 0.11$ )	0.91 ( $\pm 0.08$ )
gemini-2.5-pro	0.67 ( $\pm 0.12$ )	0.87 ( $\pm 0.09$ )	gpt-5.2	0.64 ( $\pm 0.13$ )	0.91 ( $\pm 0.08$ )
grok-4	0.6 ( $\pm 0.13$ )	0.64 ( $\pm 0.13$ )	gpt-5	0.6 ( $\pm 0.13$ )	0.81 ( $\pm 0.1$ )
o3	0.52 ( $\pm 0.13$ )	0.75 ( $\pm 0.11$ )	claude-sonnet-4.5-ext-thinking	0.49 ( $\pm 0.13$ )	0.76 ( $\pm 0.11$ )
deepseek-r1-0528	0.42 ( $\pm 0.13$ )	0.62 ( $\pm 0.13$ )	o3-mini	0.3 ( $\pm 0.12$ )	0.89 ( $\pm 0.09$ )
o4-mini	0.26 ( $\pm 0.12$ )	0.6 ( $\pm 0.13$ )	deepseek-v3.1	0.21 ( $\pm 0.11$ )	0.85 ( $\pm 0.1$ )
deepseek-reasoner	0.17 ( $\pm 0.1$ )	0.46 ( $\pm 0.13$ )	hermes-4	0.17 ( $\pm 0.1$ )	0.75 ( $\pm 0.11$ )
claude-sonnet-4-20250514	0.08 ( $\pm 0.08$ )	0.91 ( $\pm 0.08$ )	claude-opus-4-20250514	0.04 ( $\pm 0.06$ )	0.92 ( $\pm 0.08$ )
grok-3	0.02 ( $\pm 0.05$ )	1.0 ( $\pm 0.03$ )	mistral-magistral	0.02 ( $\pm 0.05$ )	0.87 ( $\pm 0.09$ )
gpt-4o	0.02 ( $\pm 0.05$ )	0.94 ( $\pm 0.07$ )	llama-3.1-405b	0.02 ( $\pm 0.05$ )	0.58 ( $\pm 0.13$ )
deepseek-v3	0.02 ( $\pm 0.05$ )	1.0 ( $\pm 0.03$ )	gpt-4.1	0.02 ( $\pm 0.05$ )	1.0 ( $\pm 0.03$ )

**Table 19.** PL3-PC accuracies and guessing-only accuracies restricted to the set of incorrect proofs.

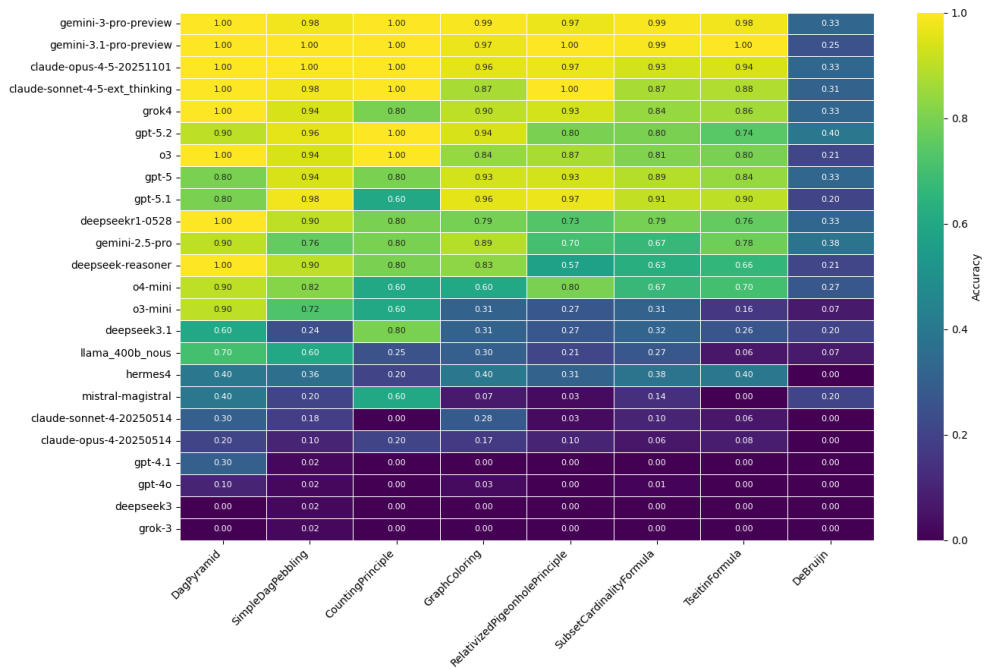


Figure 31. PL3-PC accuracy per problem type.

the relativized-pigeonhole problem family, for example, are much more challenging for such models (e.g., the accuracies of Llama-405b and Mistral Magistral fall to 21% and 3% there, respectively).

Fig. 32 shows the distribution of the various error types made by the models. We use the same error taxonomy we used for PL1-PC, with the exception of type-5 errors, which are not relevant in this setting:

1. *Type-1 errors*: The model incorrectly claims that the proof is right (false negatives).
2. *Type-2 errors*: The model incorrectly claims that the proof is wrong (false positives).
3. *Type-3 errors*: Formatting issues (e.g., responses missing expected fields).
4. *Type-4 errors*: The model correctly claims that the proof is wrong but fails to localize the error.

We see that type-2 errors dominate across all models and ability levels. Deepseek-V3, in particular, makes a stunning 247/247 type-2 errors—a false-positive error rate of 100%. The model invariably hallucinates non-existent errors. Type-1 errors, where a buggy proof is wrongly pronounced correct, are much less common, with Deepseek-Reasoner taking the lead with 28 such errors (out a total possible of 53, for an error rate of 53%), followed by Llama 405b and o4-mini (41% and 40%, respectively). Gemini

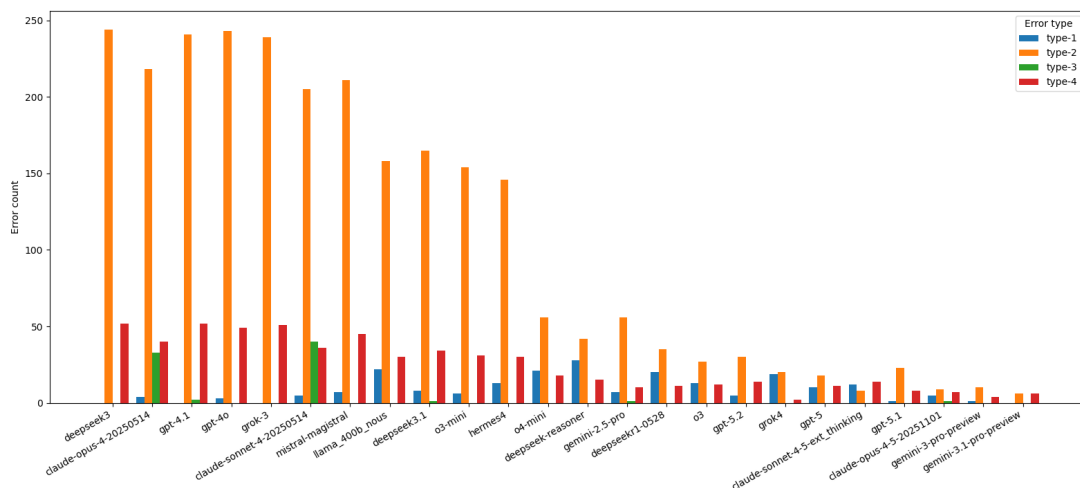
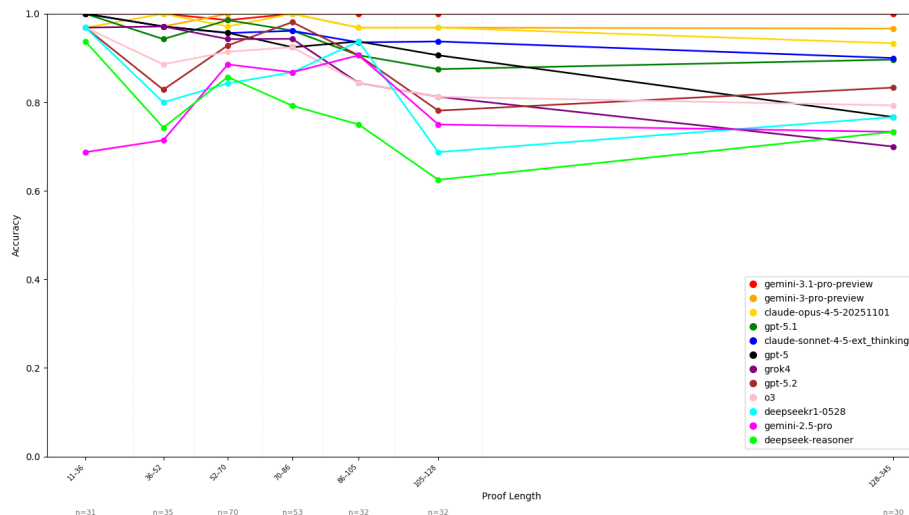


Figure 32. Error distribution across the 4 error types for PL3-PC.



**Figure 33.** PL3-PC accuracy, excluding De Bruijn problems, as a function of proof length for the top 12 models.

3.1 is the only elite model that did not make even a single type-1 error, while the two top Gemini models also made very few type-4 errors, a testament to their strong debugging skills, and specifically their ability to reliably identify subtle errors in complex reasoning chains. GPT-5.2 underperforms on this task, with a false-positive rate of 12%.

Fig. 33 plots accuracy as a function of proof length for the top 12 models. We exclude De Bruijn problems because those 15 proofs are significantly longer than the others and there are not enough of them for a robust plot. The figure shows that proof size is not a strong degrader of PL3-PC performance, at least in this particular evaluation regime of flat  $NDL_0$  proofs with no more than a few hundred steps, which translates to inputs with considerably fewer than 10K tokens.<sup>81</sup> Modern attention mechanisms show robust retrieval for context windows of that size (and even for substantially larger ones).

In conclusion, even though a few models already attain very high absolute accuracy on PL3-PC, the task remains sharply discriminative overall and exposes substantial qualitative differences in error calibration, localization, and robustness. Moreover, the observed ceiling effects are contingent on conservative corruption and scoring choices (only 53 problems were noised, in a straightforward way), and could be pushed back simply by increasing corruption density or requiring stronger justifications. (As the results show, accuracy drops for all models when evaluation is restricted to the set of incorrect proofs.) Accordingly, PL3-PC is not quite solved at present and should instead be viewed as a scalable diagnostic test of verifier-side semantic reasoning (based on entailment rather than inference rules).

## 8. A Quantitative Measure of Epistemic Stability

When a rational agent  $\alpha$  asserts a proposition, they make themselves accountable for what others can infer from it. These inferences may be made by pure deduction, but more often they are made by commonplace inferential practices in the shared space of norms that govern the use of the relevant concepts. We might say that, by asserting  $p$ ,  $\alpha$  becomes committed to additional propositions  $q_1, q_2, \dots$  that represent *epistemic obligations*, meaning that  $\alpha$  cannot deny any  $q_i$  on pain of epistemic incoherence. One of the  $q_i$  is  $p$  itself, and in that case the epistemic obligation reduces to familiar self-consistency: If  $\alpha$  asserts  $p$ , then  $\alpha$  cannot also deny  $p$  (or assert  $\neg p$ ).<sup>82</sup> But there are many other epistemic commitments. If we claim  $p = \textit{Paris is the capital of France}$ , we commit ourselves, among other

<sup>81</sup> Excluding De Bruijn problems, the average number of BPE tokens in a PL3-PC input, counting both the detailed instructions and the input proof, is 6.9K (computed with tiktoken using the cl100k\_base encoding). The median is 6.6K tokens. With De Bruijn proofs, these become 7.7K and 6.6K, respectively.

<sup>82</sup> Non-monotonic reasoning, of course, is commonplace—people often change their minds. But typically they do so upon receiving new information, at which point they might retract  $p$ . They do not maintain both  $p$  and  $\neg p$  at the same time.

things, to the propositions that France is a sovereign country and that Paris is a city in France. Unless we willfully aimed to be uncooperative communicators, it would be incoherent to deny these facts while maintaining  $p$ .

Let  $\mathcal{P}$  be any problem that may be meaningfully posed to a model  $M$ . We distinguish between the general problem  $\mathcal{P}$  and particular *instances* of  $\mathcal{P}$ .<sup>83</sup> We assume there is a prompt associated with  $\mathcal{P}$  and we write  $\pi(\mathcal{P})$  for the core part of that prompt, namely, the input-insensitive part that gives the general specification of  $\mathcal{P}$ . Even if two problems  $\mathcal{P}$  and  $\mathcal{P}'$  involve the same task, they may be formulated in incompatible ways. For instance,  $\pi(\mathcal{P})$  and  $\pi(\mathcal{P}')$  may assume different epistemic postures or operate within different epistemic contexts, e.g.,  $\pi(\mathcal{P})$  may be aimed towards truth or factual coherence whereas  $\pi(\mathcal{P}')$  may be sarcastic or adversarial, or both may be aimed towards truth but  $\pi(\mathcal{P})$  may be operating within the context of Euclidean geometry whereas  $\pi(\mathcal{P}')$  may adopt hyperbolic geometry. We say that  $\mathcal{P}$  and  $\mathcal{P}'$  are *compatible* iff no presupposition or instruction of  $\pi(\mathcal{P})$  contradicts one of  $\pi(\mathcal{P}')$ , and neither prompt provides new information to  $M$  that is not also provided by the other.<sup>84</sup> This is a symmetric relation.

We write  $M(x, y)$  to mean that given  $x$  as input, model  $M$  has generated  $y$  as the output; and we refer to  $M(x, y)$  as a *judgment*  $j$ . Now let  $\mathcal{P}$  and  $\mathcal{P}'$  be two compatible problems and let  $x$  and  $x'$  be two instances of  $\mathcal{P}$  and  $\mathcal{P}'$ , respectively. We write  $M(x, y) \vdash_N M(x', y')$  and say that  $M(x, y)$  epistemically *constrains*  $M(x', y')$  iff  $M$ 's generation of  $y$  as the answer to  $x$  creates an epistemic obligation for  $M$  to generate  $y'$  as the answer to  $x'$ , because not doing so would be epistemically incoherent. We call this a *positive* epistemic obligation. Suppose  $\mathcal{P}$  poses geography questions under a truthful system prompt  $\pi(\mathcal{P})$ , instance  $x$  asks for the capital of France, and  $M$  has generated the expected answer, Paris, as the output  $y$ . Suppose further that  $\mathcal{P}'$  asks multiple-choice trivia questions, also under a truthful system prompt, and that  $x'$  asks whether Paris is a city in France, Portugal, or Venezuela. Then  $M$  would be normatively bound to generate "France" as the answer  $y'$ . We take  $\vdash_N$  as a primitive relation for present purposes.

A *negative* epistemic obligation, written as  $M(x, y) \not\vdash_N M(x', y')$ , obtains iff  $M$ 's generation of  $y$  as the answer to  $x$  precludes it from generating  $y'$  as the answer to  $x'$ . Typically, a positive epistemic obligation creates a number of trivial negative epistemic obligations, e.g., if  $M$  generates "Paris" as the answer to question  $x$  above, then it is negatively bound from generating Portugal or Venezuela or any other country as the answer to  $x'$ . But the converse may not be true: a negative constraint from  $\mathcal{P}$  to  $\mathcal{P}'$  need not impose positive epistemic constraints on  $\mathcal{P}'$ . For instance, let  $\mathcal{P}$  and  $\mathcal{P}'$  be the proof-checking and proof-writing problems, respectively. If  $M$  claims that a proof  $D$  is incorrect, it is negatively bound from generating  $D$  as the answer to the corresponding instance of  $\mathcal{P}'$ , but this does not impose any positive obligations on  $\mathcal{P}'$ , as  $M$  is free to generate other proofs for the relevant problem. In the converse direction, however, we have positive epistemic obligations: If  $M$  generates  $D$  as the answer to an instance of  $\mathcal{P}'$ , then  $M$  is epistemically obliged to answer *yes* to the instance of  $\mathcal{P}$  that asks whether  $D$  is a correct deduction.<sup>85</sup>

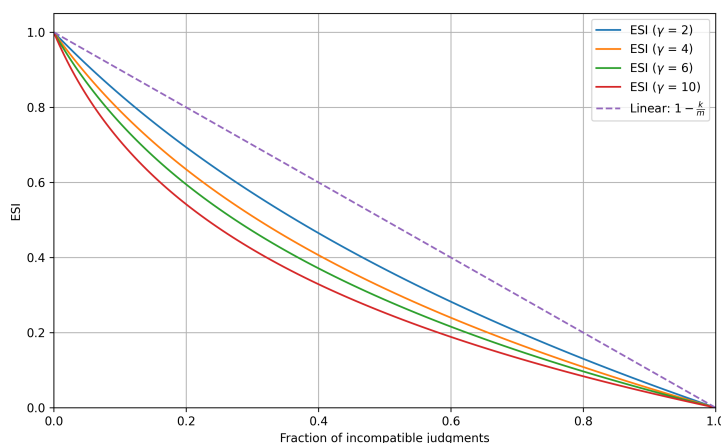
Now fix an empirical distribution  $\mathcal{D}_1$  of  $n$  observed input-output pairs  $\{(x_1, y_1), \dots, (x_m, y_m)\}$ , where each  $x_i$  is an instance of  $\mathcal{P}$  and  $M(x_i, y_i)$ ; and a corresponding distribution  $\mathcal{D}_2 = \{(x'_1, y'_1), \dots, (x'_n, y'_n)\}$  where each  $x'_j$  is an instance of a compatible problem  $\mathcal{P}'$  and  $M(x'_j, y'_j)$ . It is notationally convenient to view  $\mathcal{D}_1$  as a distribution of judgments  $M(x_1, y_1), \dots, M(x_m, y_m)$ , and likewise for  $\mathcal{D}_2$ . Define the set  $Inc_{\mathcal{D}_1}[\mathcal{D}_2]$  as follows:

$$Inc_{\mathcal{D}_1}[\mathcal{D}_2] = \{j' \in \mathcal{D}_2 \mid \exists j \in \mathcal{D}_1 . j \not\vdash_N j'\}. \quad (13)$$

<sup>83</sup> The general problem can be understood as the overall specification of the task ("Given the name of a country, respond with its population and area in square kilometers."), while an instance of the problem is a specific input to the task ("France").

<sup>84</sup> By "new" information we mean information that did not appear in  $M$ 's pretraining corpus.

<sup>85</sup> Modulo answer variation due to stochasticity. We assume here that all judgments are robust, i.e.  $M(x, y)$  means that  $y$  is a stable mode of  $M$ 's conditional distribution over outputs given  $x$ . Note, however, that in mathematical tasks, persistent answer variation when given one and the same problem instance is itself a sign of weak reasoning, indicating that the model doesn't have a stable internal representation of the solution space.



**Figure 34.** ESI curves for different values of the curvature parameter ( $\gamma$ ). The  $x$ -axis is the fraction  $|Inc_{\mathcal{D}_1}[\mathcal{D}_2]|/n$ .

In words, this set contains all judgments in  $\mathcal{D}_2$  that are precluded (negatively constrained) by, or are *incompatible with*, some judgment in  $\mathcal{D}_1$ . We can quantify the degree of  $M$ 's epistemic stability relative to these two distributions simply by computing the size of this set, dividing it by  $n$  (the size of  $\mathcal{D}_2$ ), and subtracting the result from 1. This gives us a measure of  $M$ 's epistemic stability wrt to  $\mathcal{D}_2$  and  $\mathcal{D}_1$ , written as  $ESI_{\mathcal{D}_1}[\mathcal{D}_2]$ . We drop the subscripts  $\mathcal{D}_1, \mathcal{D}_2$  when they are immaterial or obvious from the context. We refer to this quantity,  $1 - (|Inc_{\mathcal{D}_1}[\mathcal{D}_2]|/n)$ , as the *epistemic stability index* of  $M$  relative to  $\mathcal{D}_2$  and  $\mathcal{D}_1$ . Thus, *ESI* ranges smoothly from 0 to 1, with higher values obviously being preferable.

The ratio  $|Inc_{\mathcal{D}_1}[\mathcal{D}_2]|/n$  has a natural probabilistic interpretation, as the empirical probability that a randomly chosen judgment from  $\mathcal{D}_2$  is incompatible with some judgment in  $\mathcal{D}_1$ . Accordingly, 1 minus this ratio is the probability of the complementary event, namely, that a randomly chosen judgment from  $\mathcal{D}_2$  is compatible with *all* judgments in  $\mathcal{D}_1$ . We want this number to be as close to 1 as possible.

We augment this intuition with a soft log-based transformation that adds curvature reflecting nonlinear penalties for judgment inconsistencies, as well as diminishing returns from such penalties:

$$ESI_{\mathcal{D}_1}[\mathcal{D}_2] = 1 - \frac{\log \left[ 1 + \gamma \cdot \frac{|Inc_{\mathcal{D}_1}[\mathcal{D}_2]|}{|\mathcal{D}_2|} \right]}{\log[1 + \gamma]} \quad (14)$$

We still have  $ESI = 1$  when  $|Inc_{\mathcal{D}_1}[\mathcal{D}_2]| = 0$  (perfect epistemic stability) and  $ESI = 0$  when  $|Inc_{\mathcal{D}_1}[\mathcal{D}_2]| = |\mathcal{D}_2|$  (complete incoherence), but the penalty for relatively small  $|Inc_{\mathcal{D}_1}[\mathcal{D}_2]|$  grows faster than in the linear case, accentuating even small amounts of instability, while saturating smoothly towards 0 as  $|Inc_{\mathcal{D}_1}[\mathcal{D}_2]|/|\mathcal{D}_2| \rightarrow 1$ . The parameter  $\gamma$  controls the curvature and can be tuned depending on how stringent we want to be; we use  $\gamma = 5$  as a reasonable default. Figure 34 shows different ESI curves for  $\gamma \in \{2, 4, 6, 10\}$ .

We assume that  $\mathcal{D}_2$  is not chosen adversarially, with prior knowledge of the constraints in  $\mathcal{D}_1$ . Otherwise it would be possible to cherry-pick data so as to compute any desired ESI value. To avoid metric gaming, the selection of  $\mathcal{D}_2$  must be *independent* of  $\mathcal{D}_1$ , and in particular must not occur after inspecting the judgments in  $\mathcal{D}_1$ . Methodologically, this requirement is somewhat analogous to the familiar principle that test data must not be tuned to a model's performance on the training set. While this independence condition can be made fully precise (for example, by fixing the generation process for  $\mathcal{D}_2$  in advance, or by requiring  $\mathcal{D}_2$  to be sampled from a distribution specified independently of  $\mathcal{D}_1$ ), we do not need to formalize it here. For present purposes, it suffices to note that allowing  $\mathcal{D}_2$  to

Model	ESI	Model	ESI
gemini-2.5-pro	0.27	gpt-5	0.27
gpt-5.1	0.38	gemini-3.0-pro-preview	0.41
claude-opus-4-5-20251101	0.48	grok-4	0.52
o3	0.53	gemini-3.1-pro-preview	0.61
gpt-5.2	0.62		

**Table 20.** Epistemic stability values for **PL3-PW** proofs relative to individual entailment judgments for the 9 models of Table 17. A model’s *ESI* value can be understood as the probability that a randomly chosen **PL3-PW** proof written by the model is consistent with the model’s individual entailment judgments. Ideally, all of these values would be 1.

be selected in response to observed judgments in  $\mathcal{D}_1$  would undermine the intended interpretation of this stability measure.

As a concrete application of *ESI*, consider the results of Section 7.5 and let  $M$  be any of the models in Table 17. We may take  $\mathcal{D}_2$  to be the set of 300 proofs written by  $M$  in response to the **PL3-PW** prompt, or more precisely, the set  $(x, y)$  where  $x$  is an input prompt from the **PL3-PW** dataset and  $y$  is the corresponding output proof generated by  $M$ . Now consider the set  $S$  of all individual steps  $p$  from  $p_1, \dots, p_n$  that appear in any of those 300 proofs written by  $M$  ( $S$  will have thousands of elements), and let  $\mathcal{D}_1$  be the set of all entailment/SAT judgments made by  $M$  for each of those steps. In other words,  $\mathcal{D}_1$  consists of all pairs  $(x, y)$ , one for every individual step  $p$  from  $p_1, \dots, p_n$  in  $S$ , where  $x$  is an input prompt asking whether  $p$  follows from  $p_1, \dots, p_n$ , along perhaps with an estimate of the difficulty of making that judgment, and  $y$  is  $M$ ’s response.

We can then use the data in Table 17 to compute  $ESI_{\mathcal{D}_1}[\mathcal{D}_2]$ . The number of self-recognized errors becomes the size of the set  $Inc_{\mathcal{D}_1}[\mathcal{D}_2]$ , i.e., the set of all  $M$ -written proofs that are precluded by some  $M$ -made entailment judgment in  $\mathcal{D}_1$ . Plugging in the numbers from Table 17 into the defining formula (14) yields the values shown in Table 20. The results show that all top-tier models suffer from severe epistemic instability. Even for the most stable model, GPT-5.2, there is a nearly 40% chance that a randomly chosen **PL3-PW** proof written by it will be inconsistent with its own relevant entailment judgments. More than half of these elite models have *ESI* values less than 0.5, meaning that any proof written by such a model is more likely than not to disagree with the model’s corresponding entailment judgments.

At its core, *ESI* is a fraction, and therefore it is inherently asymmetric. Disregarding the log transformation, it is essentially the probability that a random draw from  $\mathcal{D}_2$  is not precluded by any draw from  $\mathcal{D}_1$ . We are thus normalizing by  $|\mathcal{D}_2|$ , not by anything symmetric across both distributions. This is to be expected, because  $ESI_{\mathcal{D}_1}[\mathcal{D}_2]$  and  $ESI_{\mathcal{D}_2}[\mathcal{D}_1]$  measure different qualitative phenomena; each has a different unit of analysis (the objects in  $\mathcal{D}_2$  and  $\mathcal{D}_1$ , respectively). For instance, with  $\mathcal{D}_1$  and  $\mathcal{D}_2$  as defined in this example,  $ESI_{\mathcal{D}_1}[\mathcal{D}_2]$  is a *proof-level* score, whereby a proof in  $\mathcal{D}_2$  is incompatible with  $\mathcal{D}_1$  iff it contains at least one step that diverges in  $\mathcal{D}_1$ . So in this case the *ESI* metric serves as an “all-steps-must-pass” aggregator. If we flip  $\mathcal{D}_1$  and  $\mathcal{D}_2$ , then  $ESI_{\mathcal{D}_2}[\mathcal{D}_1]$  would become a *step-level* score that would address a very different question: what fraction of the model’s entailment judgments survive contact with the corresponding proofs? Both quantities may be of interest, but no matter what  $\mathcal{D}_1$  and  $\mathcal{D}_2$  are, *any* low values for  $ESI_{\mathcal{D}_2}[\mathcal{D}_1]$  are a source of concern.

In general, epistemic stability is not an absolute scalar property. It is defined with respect to a reference distribution, for which there is no universally correct choice. Definition (14) is general, but genuinely different stability concepts can be singled out by instantiating the distributions in different ways. For example, above we took  $\mathcal{D}_2$  to be the set of all 300 **PL3-PW** outputs by  $M$ , but we could instead consider the subset comprising only the *incorrect* outputs generated by  $M$  (while leaving  $\mathcal{D}_1$  unchanged). Doing so would pick a different measure of epistemic stability, one that we might call *error-triggered ESI*, or  $ESI^{ET}$  for short. (We do not report  $ESI^{ET}$  values here for **PL3-PW**, but it suffices to note that they are very low, e.g., 0.14 for Gemini 3.1 and 0.38 for GPT-5.2.) The first version of *ESI* can be viewed as a global or generic measure of epistemic stability, while the second is conditioned on

errors. The difference between the two is roughly analogous to the difference between blood pressure at rest vs blood pressure under stress. They measure two different aspects of epistemic stability. An agent with high error-triggered stability makes errors that are part of a stable internal theory, even if that theory is wrong. In other words, the agent may be wrong but they are wrong in a unified and stable way. This is common for humans (disregarding trivial performance errors). By contrast, low error-triggered stability means that the agent's mistakes are not stably believed, arising instead from breakdowns in deployment, control, or integration. This appears to be common for language models, which often have the competence to make the right local judgments (such as micro-SAT entailments) but fail to recruit that competence during more complex and global tasks (such as proof writing).

It may be noted that epistemic stability is not unequivocally desirable. In particular, the capacity for self-correction and epistemic stability appear to pull in opposite directions. Low stability values indicate poor integration of different types of competence, but they also suggest that the model possesses latent knowledge that can be recruited by external scaffolding or iterative self-checking and verification. High stability values, by contrast, indicate that the model's mistakes are part of a stable internal reasoning pattern and are thus less amenable to agentic self-correction. So there seems to be a tradeoff: low epistemic stability is bad for unified reasoning but good for self-aided corrigibility, while high epistemic stability is good for unified and internally stable reasoning but not for self-repair. While this tension is generally real, in practice epistemic stability is the more valuable primitive; we need systems that are stable, self-consistent, and predictable across contexts. At any rate, the tension is only with self-correction, not with correction in general. From an engineering perspective, one does not need a single model to, e.g., both generate and verify. Corrections can be made by external verifiers or critic agents rather than by relying on the same model to repair its own inconsistencies.

## 9. PL<sub>4</sub>

### 9.1. Proof Writing in Hilbert Systems

This division contains 250 proof-writing problems in the Hilbert calculus for propositional logic. Because writing proofs in a Hilbert system is challenging, we have split this division into two tasks:

1. the main **PL4-PW** task, consisting of 200 problems that are shorter and easier on average than **PL1-PW** items;
2. a challenge set, **PL4-PW-c**, consisting of 50 problems randomly sampled from longer **PL1-PW** problems.

The Hilbert calculus is a flat proof system that does not admit conditional subproofs: a proof is just a list of formulas, each of which is an instance of an axiom schema or else follows from previous steps by a single inference rule, modus ponens. This flat linear structure makes Hilbert systems primarily of metatheoretical interest (e.g., for studying soundness and completeness), because the lack of assumption scope and the pared-down rule set make it easier to prove theorems about these systems than within them. But for benchmarking purposes, these same factors make Hilbert proofs substantially harder to write than proofs in natural deduction.

Formulas here are defined exactly as in NDL, built up inductively from propositional atoms and logical connectives. In the variant we use, which is the most common formulation of the Hilbert calculus, the only two primitive connectives are negation and implication. Disjunctions, conditionals, and biconditionals are desugared using only these two connectives. In particular,  $p \vee q$  is written as  $\neg p \Rightarrow q$ ,  $p \wedge q$  is written  $\neg(\neg p \Rightarrow \neg q)$ , and  $p \Leftrightarrow q$  as  $\neg((p \Rightarrow q) \Rightarrow \neg(q \Rightarrow p))$ .

Conditional elimination (modus ponens, also known as "detachment") is the only inference rule; it derives  $q$  from  $p \Rightarrow q$  and  $p$ . In addition, any proof step may infer any instance of an axiom schema (a logical formula template). For example, Axiom 1 of the calculus is  $(p \Rightarrow (q \Rightarrow p))$ , where  $p$  and  $q$  are metavariables ranging over arbitrary formulas. Any formula that *matches* this template is an axiom, such as  $(A \Rightarrow (B \Rightarrow A))$  or  $(\neg A \Rightarrow ((B \vee C) \Rightarrow \neg A))$ .

The standard version of the Hilbert calculus that we use formalizes a classical logic of implication and negation with only three propositional axiom schemas:<sup>86</sup>

- Axiom 1:  $(p \Rightarrow (q \Rightarrow p))$
- Axiom 2:  $(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$
- Axiom 3:  $(\neg p \Rightarrow \neg q) \Rightarrow ((\neg p \Rightarrow q) \Rightarrow p)$

A derivation of a target tautology  $p$  is a sequence of formulas  $p_1, \dots, p_n = p$ , where each  $p_i$  is an axiom instance or else follows by modus ponens from two earlier formulas. This format is sufficient in principle, but, for convenience, we name each line and provide a justification either by citing an axiom or by listing the arguments to modus ponens. For example, the following is a proof produced by o4-mini for the tautology  $((\neg A \Rightarrow A) \Rightarrow A)$ . Each axiom is implemented as a primitive method in Athena (Athena Language Team 2025, p. 103), which matches the input sentence to the respective axiom schema or reports that no match is possible. In this way the proof becomes fully verifiable using Athena.

```
{
p0 := axiom-1 on (~A ==> ((~A ==> ~A) ==> ~A));
p1 := axiom-2 on ((~A ==> ((~A ==> ~A) ==> ~A)) ==> ((~A ==> (~A ==> ~A)) ==> (~A ==> ~A)));
p2 := ((~A ==> (~A ==> ~A)) ==> (~A ==> ~A)) BY mp on p1, p0;
p3 := axiom-1 on (~A ==> (~A ==> ~A));
p4 := (~A ==> ~A) BY mp on p2, p3;
p5 := axiom-3 on ((~A ==> ~A) ==> ((~A ==> A) ==> A));
p6 := ((~A ==> A) ==> A) BY mp on p5, p4
}
```

We also permit the use of premises, treating them essentially as axioms. As a result, we can pose problems of the form “prove that  $p$  follows from  $\{p_1, \dots, p_n\}$ ” in addition to pure-tautology problems of the form “prove  $p_1 \Rightarrow \dots \Rightarrow p_n \Rightarrow p$ .” This is an important change that allows models to perform suppositional reasoning and emulate natural deduction, yielding a flexible knob controlling problem difficulty. As we explain later, it also enables some interesting experiments gauging the ability of models to simplify proof construction in this system by intelligently exploiting the famous deduction theorem Mendelson (2010)[p. 28].<sup>87</sup>

**PL4-PW**, the (easy) 200-problem task, was constructed as follows: (a) 20 problems were hand-picked simple tautologies that appear as standard textbook exercises, such as  $(A \Rightarrow (\neg A \Rightarrow B))$ ,  $(A \Rightarrow \neg \neg A)$ , and  $(A \vee \neg A)$ ; and (b) 180 randomly selected problems from the same distribution from which **PL1-PW** problems were sampled. We targeted short problems (mean length 46 characters) to keep the task feasible. Any problem using logical connectives other than negation and implication was converted to an equivalent formula using only negation and implication, as described above. Only 15 out of the 200 problems are pure tautologies; the rest have at least one premise.

The size distributions for problems in **PL4-PW** and **PL4-PW-c**, measured as the combined AST sizes of the original premises and goals, are shown in Figure 35. Table 21 summarizes the mean and max numbers of atoms, negations, conjunctions, disjunctions, conditionals, and biconditionals in these datasets.

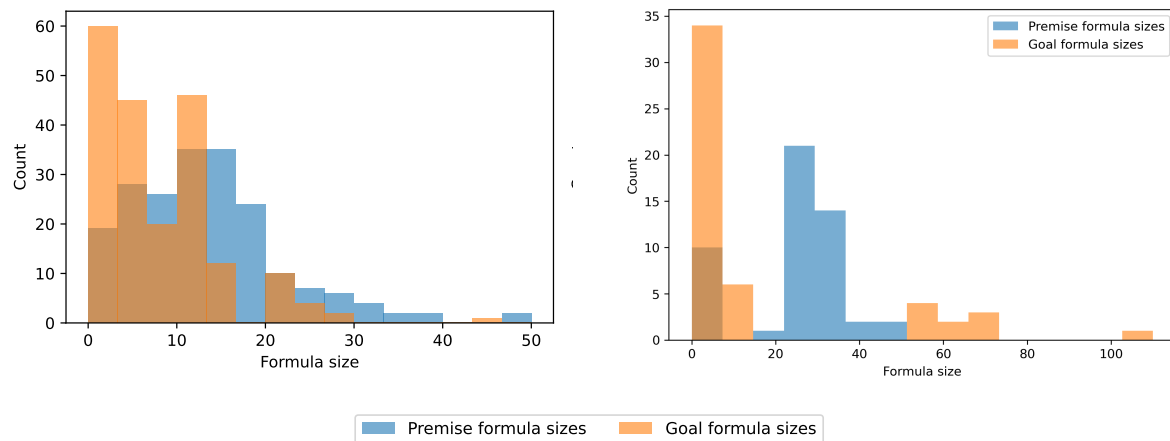
The average problem size in **PL4-PW** is about 21, compared to about 28 for **PL1-PW**. In the challenge dataset **PL4-PW-c**, the average problem size is  $\approx 40$ .

## 9.2. PL4-PW Results

We evaluated model performance under two settings: (1) a strict setting in which the model is required to produce a proof that can be transformed into an Athena-verifiable proof without any

<sup>86</sup> The first two axiom schemas are known as  $K$  and  $S$ , respectively, as they correspond precisely to the  $K$  and  $S$  combinators of combinatory logic Hindley et al. (1972), namely  $\lambda x . \lambda y . x$  and  $\lambda f . \lambda g . \lambda x . (f x) (g x)$ . To see the correspondence of the latter with Axiom 2, read  $f$  as a function of type  $p \Rightarrow (q \Rightarrow r)$ ,  $g$  as a function of type  $p \Rightarrow q$ , and  $x$  as a variable of type  $p$ . We then we have  $(f x) : q \Rightarrow r$  and  $(g x) : q$ , so that  $(f x) (g x)$  is of type  $r$ , corresponding to the tail occurrence of  $r$  in Axiom 2. Dropping Axiom 3 and keeping only  $K$  and  $S$  gives rise to the classical logic of minimal implication.

<sup>87</sup> The deduction theorem, independently discovered by Tarski (1930) and Herbrand (1930), states that if there is a Hilbert proof of  $p$  from premises  $\{p_1, \dots, p_n, p_{n+1}\}$ , then there is a Hilbert proof of  $(p_{n+1} \Rightarrow p)$  from  $\{p_1, \dots, p_n\}$ .



**Figure 35.** Problem size distributions for **PL4-PW** (left) and **PL4-PW-c** (right).

post-processing; and (2) a lenient setting in which we apply the following limited programmatic corrections to superficial syntax issues and certain formatting errors in applications of modus ponens:

- We use a parsing algorithm that can repair certain formulas with unbalanced parentheses by adding or removing parentheses at the right end of the formula to restore balance; this recovers some formulas that would otherwise be rejected by the Athena parser.
- We allow different parenthesization styles, including minimal parenthesization where connective application is inferred according to Athena’s precedence and associativity rules.
- We repair missing or superfluous semicolons and other simple syntax errors that are readily fixable by programmatic search-and-replace.
- We correct flipped argument order in applications of detachment when the swap yields a valid rule application.

The above can be seen as a limited amount of proof-checking instrumentation (compared to the more extensive interventions in NDL).

**PL4-PW** results for both modes are shown in Table 22. Gemini 3.1 is in a class of its own in this task, far ahead of all other models.

The DI for **PL4-PW** is 0.13 (for lenient-mode results). The lower DI is due to the overall lower performance.

For the challenge set **PL4-PW-c**, we only evaluated a selection of elite models. The results are shown in Table 23. On this task, Gemini 3.1 solved 6 out of 50 problems, GPT-5.2 solved 3, Gemini 3 solved 1, and all other models failed to produce a single correct proof. Evaluating additional weaker models on this task would not be meaningful.

In general, writing proofs in a Hilbert calculus is a challenging task that requires high ingenuity and exceptional skill and precision in symbolic manipulation and pattern matching. This is borne out

Formula Feature	Mean	Max
Size	20.97	52
Atoms	8.24	19
Negations	7.17	23
Conjunctions	0.00	0
Disjunctions	0.00	0
Conditionals	5.55	15
Biconditionals	0.00	0

**Table 21.** **PL4-PW** Problem Statistics. Size = total number of nodes across all formulas; Atoms = number of propositional variables; Negations, Conjunctions, Disjunctions, Conditionals, Biconditionals = counts of respective logical connectives.

Model	Strict	Lenient	Model	Strict	Lenient
gemi-3.1-pro-preview	<b>0.55</b> ( $\pm 0.04$ )	<b>0.71</b> ( $\pm 0.03$ )	gpt-5.2	0.26 ( $\pm 0.03$ )	0.39 ( $\pm 0.03$ )
gemi-3.0-pro-preview	0.20 ( $\pm 0.03$ )	0.23 ( $\pm 0.03$ )	grok-4	0.10 ( $\pm 0.02$ )	0.16 ( $\pm 0.03$ )
claude-opus-4-5-20251101	0.14 ( $\pm 0.02$ )	0.14 ( $\pm 0.02$ )	deepseek-reasoner	0.12 ( $\pm 0.02$ )	0.14 ( $\pm 0.02$ )
gpt-5	0.11 ( $\pm 0.02$ )	0.14 ( $\pm 0.02$ )	gpt-5.1	0.07 ( $\pm 0.02$ )	0.14 ( $\pm 0.02$ )
gemi-2.5-pro	0.08 ( $\pm 0.02$ )	0.12 ( $\pm 0.02$ )	o3	0.06 ( $\pm 0.02$ )	0.12 ( $\pm 0.02$ )
claude-sonnet-4.5-ext-thinking	0.12 ( $\pm 0.02$ )	0.12 ( $\pm 0.02$ )	deepseek-r1-0528	0.09 ( $\pm 0.02$ )	0.10 ( $\pm 0.02$ )
o4-mini	0.06 ( $\pm 0.02$ )	0.07 ( $\pm 0.02$ )	o3-mini	0.05 ( $\pm 0.02$ )	0.07 ( $\pm 0.02$ )
hermes-4	0.04 ( $\pm 0.01$ )	0.04 ( $\pm 0.01$ )	claude-sonnet-4-20250514	0.04 ( $\pm 0.01$ )	0.04 ( $\pm 0.01$ )
claude-opus-4-20250514	0.04 ( $\pm 0.01$ )	0.04 ( $\pm 0.01$ )	deepseek-v3.1	0.04 ( $\pm 0.01$ )	0.04 ( $\pm 0.01$ )
grok-3	0.03 ( $\pm 0.01$ )	0.03 ( $\pm 0.01$ )	deepseek-v3	0.03 ( $\pm 0.01$ )	0.03 ( $\pm 0.01$ )
llama-3.1-405b	0.03 ( $\pm 0.01$ )	0.03 ( $\pm 0.01$ )	gpt-4o	0.02 ( $\pm 0.01$ )	0.02 ( $\pm 0.01$ )
gpt-4.1	0.01 ( $\pm 0.01$ )	0.01 ( $\pm 0.01$ )	mistral-magistral	0.01 ( $\pm 0.01$ )	0.01 ( $\pm 0.01$ )

**Table 22. PL4-PW accuracies in strict and lenient modes. Top accuracies in both modes are shown in bold.**

Model	Accuracy
gemi-3.1-pro-preview	<b>0.12</b>
gpt-5.2	0.06
gemi-3.0-pro-preview	0.02
grok-4	0.00
claude-opus-4-5-20251101	0.00
deepseek-reasoner	0.00
gpt-5	0.00
gpt-5.1	0.00

**Table 23. PL4-PW-c accuracies for top-performing models.**

Model	Number of Proof Steps				
	$\varphi_1$	$\varphi_2$	$\varphi_3$	$\varphi_4$	$\varphi_5$
gemi-3.1-pro-preview	9	25	19	55	59
gpt-5.2	25	25	19	-	-
gemi-3.0-pro-preview	-	25	19	-	-
grok-4	-	-	-	-	-
claude-opus-4-5-20251101	-	25	19	-	-
deepseek-reasoner	-	29	19	-	-
gpt-5	-	-	19	-	-
gpt-5.1	-	89	19	-	-
gemi-2.5-pro	-	-	-	-	-
o3	-	-	19	-	-

$$\varphi_1 : (A \Rightarrow \neg\neg A)$$

$$\varphi_2 : (\neg A \Rightarrow (A \Rightarrow B))$$

$$\varphi_3 : ((\neg A \Rightarrow \neg B) \Rightarrow (B \Rightarrow A))$$

$$\varphi_4 : ((\neg A \Rightarrow (\neg B \Rightarrow C)) \Rightarrow (\neg(\neg A \Rightarrow B) \Rightarrow C))$$

$$\varphi_5 : ((\neg A \Rightarrow (\neg B \Rightarrow C)) \Rightarrow (\neg C \Rightarrow (\neg B \Rightarrow A)))$$

**Table 24. Performance of leading models on 5 simple tautologies in PL4-PW. The number of steps of a successful proof is shown for each model and formula; a dash indicates failure to produce a correct proof.**

by the poor overall performance of current models even on very simple tautologies. Table 24 shows the number of steps in a successful proof for each model and formula, for 5 of the simplest problems in the dataset. A dash “-” indicates that the model failed to produce a correct proof. Gemini 3.1 was the only model to solve all 5 problems.

### 9.2.1. Error Analysis

Errors in this task are grouped into the following categories:

- *Failed MP*: The model applied detachment incorrectly.

- *Failed Axiom*: The model used an axiom improperly. These errors are essentially axiom hallucinations, where the model falsely claims that a sentence is an instance of an axiom.
- *Undefined/Hallucinated Intermediate Step name*: This occurs when an intermediate-step name  $p_i$  is quoted in an application of modus ponens without being introduced in a previous step.
- *Wrong Conclusion*: The proof reaches a conclusion other than the goal.
- *No Proof*: The model did not generate a proof. This occurs in two ways: either the model fails to generate any response after multiple API calls for a given problem, or the model returns a response that does not contain a proof. (This is especially common for o3, occurring in 102 out of the 200 problems.)
- *Syntax Error*: The model returned a proof with a syntax error that our automated repairs could not fix.

The distribution of these errors in **PL4-PW** (under lenient proof checking) is shown in Figure 36. Incorrect applications of modus ponens in **PL4-PW** fall into one of three types:

1. the first argument is not a conditional formula (a type error of sorts);
2. the rule is applied to two formulas  $p \Rightarrow q$  and  $r$  where  $r \neq p$  (also a type error); and
3. at least one of the two arguments is not in the assumption base (a strictly logical error).

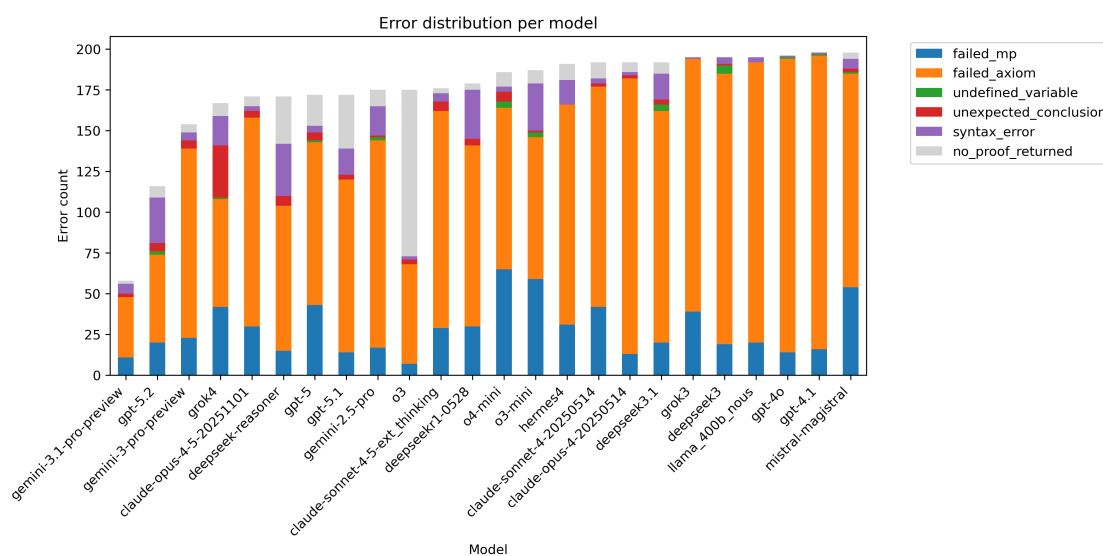
Most failed-mp errors made by the strongest models are of type 2.

### 9.2.2. Proof Length Analysis

Success rates in this task correlate strongly both with problem size and with the length of the generated proofs. Figure 37 shows model accuracies as a function of problem size, defined as the total AST size of the premises and goal. Performance degrades rapidly as problem size increases, even though problems in this task are quite shorter than those in **PL1-PW**.

Successful proofs for this task are generally short. The shortest proofs for the vast majority of solved problems have fewer than 30 lines.

Statistics on the lengths of all proofs written by all models for **PL4-PW**, similar to those in tasks like **PL2-PW**, are shown in Figure 38. For each model  $M$ , the figure shows the distribution of length over the set of all proofs written by  $M$  (blue), as well as the distribution of length over the set of all correct proofs written by  $M$  (green). Again, for most models the green distributions (correct proofs only) are narrower and shifted left compared to the blue distributions (all proofs), again showing that correct proofs tend to be shorter. However, some strong models (gpt-5.2, gpt-5.1, gpt-5, and even grok-4, deepseek-reasoner, and o3) generate correct proofs of higher mean length than that of all their



**Figure 36.** The distribution of error types in **PL4-PW** under lenient proof checking, for the first error that caused a proof failure (or the failure to return a proof).

proofs. Several models write extremely long incorrect proofs (Gemini 2.5 produced the longest, a proof with 2045 lines), suggesting that these models may generate long, verbose proof attempts when they are unable to solve a problem.

We also compared a set of strong models in terms of the length of their successful proofs, by calculating the average proof length across the problems that all of these models managed to solve (under lenient settings): gpt-5.2, grok-4, gpt-5.1, gemini-3.0-pro-preview, deepseek-reasoner, gemini-2.5-pro, claude-opus-4-5-20251101, and claude-sonnet-4.5-ext-thinking. We found that gpt-5 produced the shortest proofs across the 17 problems that all of these models solved, with an average proof length of 8.4 steps, while gemini-2.5-pro produced the longest proofs, with an average of 9.4 steps. Remarkably, most models found a proof of the same length for each problem, suggesting that the models converged on the same proof.

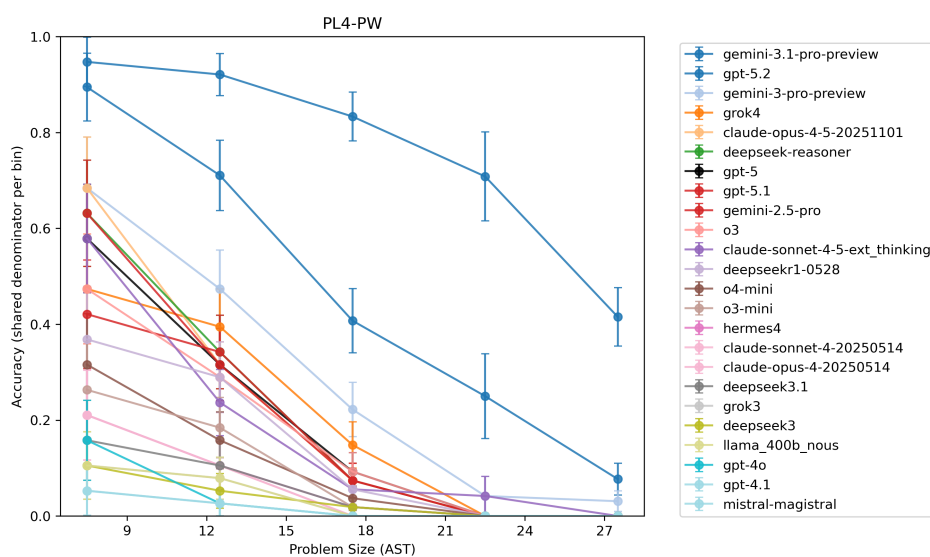


Figure 37. PL4-PW accuracy as a function of problem size.

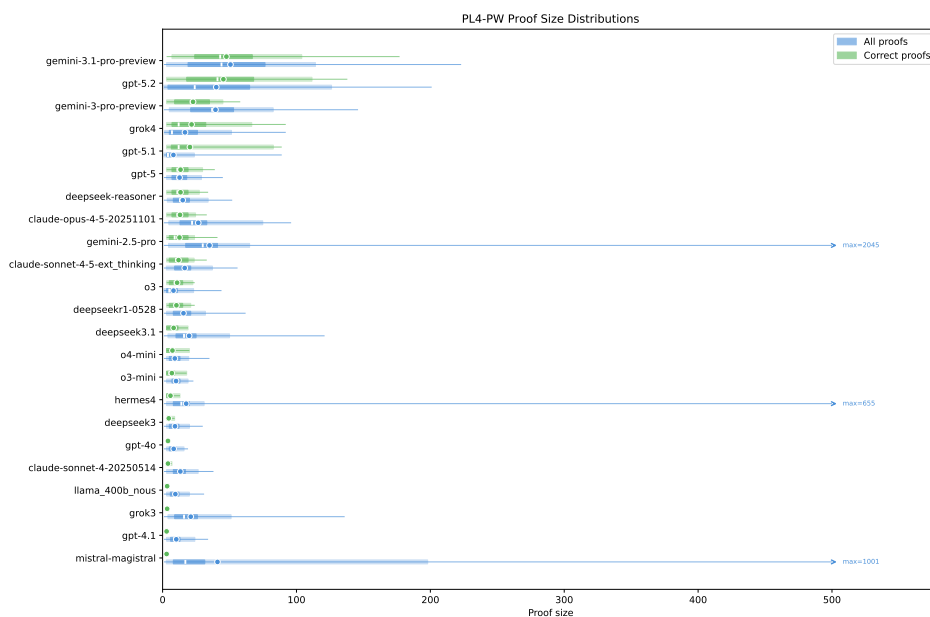


Figure 38. PL4-PW proof-size distributions per model: all proofs (blue) vs correct proofs (green); min-max line, 5-95% band, IQR box, median tick, mean dot.

### 9.2.3. Proof Similarity Analysis

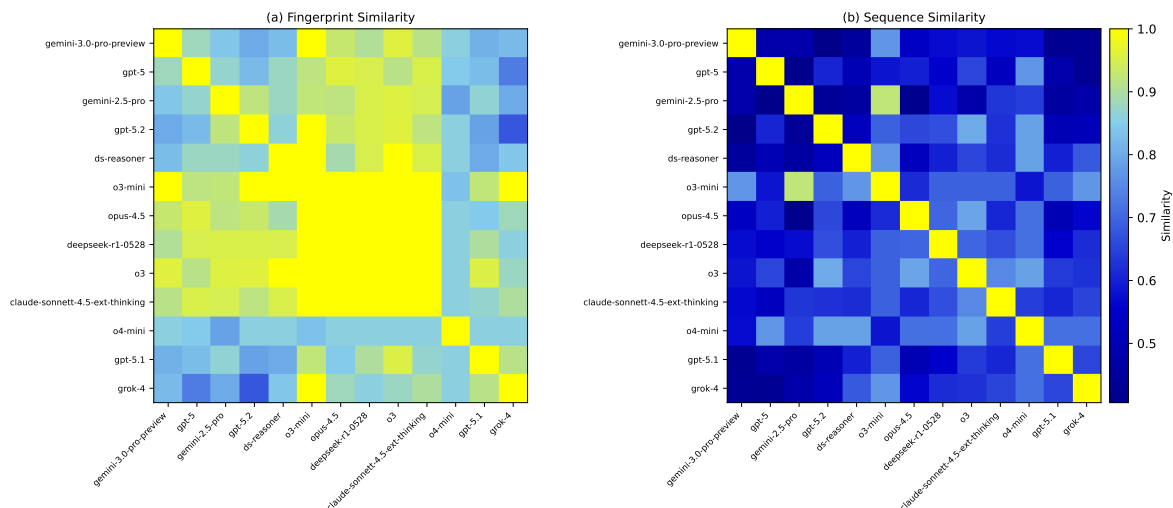
To further investigate the relationship between proof length and correctness, we computed the average proof length ratio of correct vs. incorrect proofs for each model as follows. We first identified all problems for which at least 3 models succeeded and at least 3 models failed, where the failures were required to be due to errors in the proof rather than failures to return a proof. For each of these problems, we computed the shortest correct proof length and ranked models according to their average ratio of correct proof lengths to shortest correct proof lengths. We then computed the average ratio of incorrect proof lengths to shortest correct proof lengths for each model across these problems. The results are shown in Table 25. Note that neither gemini-3.1-pro-preview nor gpt-5.2 is included because they did not have at least 3 incorrect proofs among these problems, and a number of weaker models are not included because they did not have at least 3 correct proofs among these problems. As a general trend, most models tend to write correct proofs that are similar in length to the shortest correct proof, with the exception of gpt-5.1 and grok-4, which tend to write longer correct proofs. Among incorrect proofs, some models tend to write much shorter proofs. Other models tend to write much longer proofs, with gemini-3.0-pro-preview having the largest ratios; it could be said that these models tend to persist in proof writing even when they “know” that the proof is not succeeding.

As first noted in connection with Table 24, we observed that models usually return proofs of identical length for the same problem. This is often an indication that the models have converged on the same proof. To further investigate this, we computed the pairwise similarity between model proofs as follows. First, we defined the “fingerprint” of a proof as the counts of each axiom and applications of detachment in the proof. According to this measure, two proofs are identical iff they have the same length, the same counts of each axiom application, and the same number of detachment applications. Second, we defined the “sequence” of a proof as the order in which the axioms and detachment applications appear in the proof. According to this measure, the proofs must have identical fingerprints and, additionally, the same precise order of application of the axioms and detachment.

Figure 39 shows pairwise similarity heatmaps according to the fingerprint and sequence measures for PL4-PW. For each pair of models, *fingerprint similarity* is the percentage of commonly solved problems for which the two successful proofs have the same fingerprint (counts of each axiom and detachment applications), while *sequence similarity* additionally requires the same order of axiom and detachment applications. We include a model in the heatmap only if it shares at least 10 commonly solved problems with every other included model (models are ordered by their performance for inclusion, although this ordering does not affect the final inclusion list). As the heatmaps show, the models demonstrate a remarkable degree of convergence to essentially the same proofs. Proofs that

Model	Avg $\ell/\ell_{\min}$ (C)	Avg $\ell/\ell_{\min}$ (I)	Ratio (I/C)
grok-3	1.00	0.93	0.93
deepseek-v3.1	1.00	0.98	0.98
claude-sonnet-4-20250514	1.00	0.94	0.94
hermes-4	1.00	0.82	0.82
o3-mini	1.03	0.62	0.60
o3	1.03	1.01	0.98
o4-mini	1.04	0.60	0.57
claude-sonnet-4.5-ext-thinking	1.06	0.87	0.82
deepseek-r1-0528	1.06	0.87	0.83
claude-opus-4-5-20251101	1.10	1.17	1.06
deepseek-reasoner	1.11	0.60	0.54
gpt-5	1.13	0.70	0.62
gemini-3.0-pro-preview	1.19	1.72	1.45
gemini-2.5-pro	1.19	3.41	2.86
grok-4	1.34	1.20	0.89
gpt-5.1	1.38	0.59	0.43

**Table 25.** Proof length ratio analysis on PL<sub>4</sub>-PW problems of moderate difficulty (problems with  $\geq 3$  correct and  $\geq 3$  incorrect proofs among models). For each model we show mean  $\ell/\ell_{\min}$  ratios, where  $\ell$  is proof length and  $\ell_{\min}$  is the length of the shortest correct proof for that problem. C (I) = correct (incorrect) proofs. Rows are ranked by avg(C), lower is better.



**Figure 39.** Pairwise proof similarity between models on PL<sub>4</sub>-PW problems. Models are grouped by hierarchical clustering based on fingerprint similarity. Yellow indicates higher similarity; dark blue indicates lower similarity.

match in fingerprint are often identical up to some reordering of the applications of different axioms and MP.

The results of this section show that even when the underlying problems are unchanged from **PL1-PW** (and are, in fact, drawn from the easier end of that distribution), changing the proof formalism to one that does not mirror human reasoning increases the level of proof-writing difficulty by an order of magnitude. **PL4-PW** thus stands apart from other propositional-logic tasks in **PROOFGRID**, all of which are rooted in natural deduction. Writing Hilbert proofs does not test human-style reasoning. Instead, it probes the ability to compile proofs into a very sparse calculus that is essentially a machine language for deduction. It places much higher pressure on symbolic manipulation discipline than on high-level semantic reasoning about the problem. As such, **PL<sub>4</sub>** is complementary to the other **PROOFGRID** divisions, providing a compelling additional axis: proof-writing skill under a deliberately unforgiving and syntactically brittle formalism.

However, there is an interesting direction for future work that would go beyond raw symbolic muscle to test human-like meta-reasoning: study whether models are able to exploit the deduction theorem in Hilbert systems, instead of attempting every proof directly from the 3 axioms. For example, to prove implication transitivity,  $(A \Rightarrow B) \Rightarrow (B \Rightarrow C) \Rightarrow (A \Rightarrow C)$ , a human would typically not search for a pure (premise-free) Hilbert derivation from first principles. Instead, they would first derive  $C$  from the temporary assumptions  $A$ ,  $A \Rightarrow B$ , and  $B \Rightarrow C$ , and then repeatedly apply the deduction theorem to discharge these assumptions and obtain a pure Hilbert proof of the conditionalized tautology. Since this discharge process is constructive (and can be carried out by a simple algorithm), it suggests a natural controlled experiment: compare model performance on the premise-based Hilbert proofs in **PL4-PW** with their performance on the corresponding conditionalized (premise-free) versions. This would address the question of whether a model can solve a conditionalized Hilbert problem by first solving the easier premise-based version and then mechanically compiling the solution into a pure Hilbert proof. A model that can operationalize the deduction theorem in roughly the same way a human reasoner would is expected to degrade only moderately between the two settings, whereas a sharp drop would suggest that it is not leveraging a natural meta-level proof compilation strategy. Although we do not pursue this experiment here, it points to a promising way in which Hilbert systems may probe not just low-level proof synthesis but also strategic reasoning about proofs.

### 9.3. Equational Logic

Equational reasoning sits at the heart of mathematics, and by extension, of science and engineering. Work in all of these fields routinely involves equations between apparently different expressions that refer to the same entity. Engineers use equational reasoning to show that two circuit designs are functionally identical, physicists to track conserved quantities, and compiler engineers to verify that back-end optimizations yield identical program semantics. In computer science, the use of rewrite rules to reason about equations constitutes a Turing-complete model of computation, forming the foundation of functional programming and playing a key role in program verification. Any model aspiring to master these domains must internalize equational reasoning as a core competency.

Equational reasoning offers a particularly attractive testing ground for AI models because it is almost entirely language-independent. Proofs do not need to rely on the specialized syntactic rules of any particular formal system, but can instead be written as simple sequences of steps where each step is just a term annotated with the equational axioms used to derive it from the preceding term.<sup>88</sup> Such representations are both rigorous and machine-checkable, while avoiding the burden of ensuring adherence to the idiosyncrasies of a formal language. Despite this structural simplicity, equational reasoning is remarkably rich and powerful, capable of encoding subtle and sophisticated arguments. Indeed, in the general case equational reasoning is undecidable, which underscores that the expressive reach of this style of reasoning is more than adequate for probing the limits of AI models.

Reasoning about chains of identities requires both domain-independent syntactic skill (carrying out manipulations correctly) and domain-specific semantic insight (grasping the meaning of equivalence in a particular theory). In this paper we focus on the former. Using an uninterpreted signature, we introduce three tasks that evaluate a model's general ability to recognize and apply identity transformations. Because complex terms are inherently structured as recursive syntax trees, these tasks also probe a model's ability to process hierarchically organized data with rigor and precision.

Specifically, we introduce EQ<sub>1</sub>, a dataset containing 1K purely equational (unconditional) proofs. Each example can be viewed as a pair that includes two main components:

- a set of equational axioms  $\{E_1, \dots, E_m\}$ , where, for easy reference, all  $m$  equational axioms are given unique names:  $E_1, \dots, E_m$ ; and
- a derivation of an identity  $s = t$  from those axioms. The structure of derivations is described below.

Each axiom is of the form  $\forall x_1, \dots, x_n. l = r$ , where the  $x_i$  are variables and  $l, r$  are standard first-order terms built from variables, constants, and function symbols of nonzero arity, e.g.,  $g(a, f(Y))$ . Variable names start with upper-case letters, while function symbols and constants start with lower-case letters. We use  $a, b, c, d$  and  $e$  as constants. Unary function symbols are  $f, f_1, \dots$ ; binary symbols are  $g, g_1, g_2, \dots$ ; ternary symbols are  $h, h_1, \dots$ ; and quaternary function symbols (of arity 4) are  $r, r_1, \dots$ . We don't use any function symbols with arity greater than 4.

If  $op$  is a function symbol of arity  $n$ , the abstract syntactic structure of a term of the form  $op(t_1, \dots, t_n)$  is captured by a *tree* with the function symbol  $op$  at the root and with the trees corresponding to  $t_1, \dots, t_n$  as its children, ordered from left to right. Constants and variables always appear at the leaves. Viewing terms as trees in this way, *positions* within a term are represented using Dewey notation. Thus, if  $t = g(f(a), g(X, f_2(Y)))$ , position  $[]$  corresponds to the subterm of  $t$  at the root (i.e.,  $t$  itself), position  $[1]$  corresponds to  $f(a)$ ,  $[1, 1]$  corresponds to the constant (leaf)  $a$ ,  $[2, 1]$  corresponds to  $X$ ,  $[2, 2]$  to  $f_2(Y)$ , and so on. For a term  $t$  and valid position  $p$  in  $t$ ,  $sub(t, p)$  denotes the subterm of  $t$  at  $p$ . We write  $Dom(t)$  for the set of all valid positions in  $t$ .

A substitution  $\theta$  is a function from variables to terms, e.g.,  $\theta = \{X \mapsto f(a, Y), Z \mapsto b\}$ . We write  $\theta(t)$  to denote the result of *applying*  $\theta$  to a term  $t$ , namely, the term obtained from  $t$  by replacing every

<sup>88</sup> Of course, the terms must still be written in formal notation, such as  $plus(succ(X), times(a, zero))$ . But this requires nothing more than the standard convention of a function symbol followed by parentheses enclosing comma-separated arguments. This notation is both universal and simple, and is already extensively represented in the generic corpora on which language models are pretrained.

variable  $v$  in the domain of  $\theta$  with the term  $\theta(v)$ . Thus, for the given example,  $\theta(g(Y, g(X, Z))) = g(Y, g(f(a, Y), b))$ . We say that a term  $t$  matches a term  $s$ , or that  $t$  is an instance of  $s$ , iff there exists a substitution  $\theta$  such that  $t = \theta(s)$ , in which case we say that  $t$  matches  $s$  “under”  $\theta$  (or wrt  $\theta$ ).

A proof of  $s = t$  from a set of equational axioms  $\mathcal{E}$  is a sequence of  $n > 0$  steps of the following form:

$$\begin{aligned} s &= s_0 \\ &= s_1 \text{ by } \dots \\ &= s_2 \text{ by } \dots \\ &\vdots \\ &= s_n \text{ by } \dots \end{aligned}$$

where  $s_n$  is syntactically identical to  $t$ . Such a proof can be understood as a chain of identities leading from  $s$  to  $t$ . Each step is justified by citing one or more of the axioms in  $\mathcal{E}$ , listed after the **by** keyword. The following is a sample equational proof of 5 steps that derives the identity  $g(h(c, a, c), h(a, c, a)) = g(g_1(g_4(f_1(f_5(f_3(a))), d), e), g_2(a, c))$  from a set of 6 equational axioms. (Universal quantifiers are omitted when listing the equational axioms; all variables are assumed to be universally quantified.)

```
- Axioms :
E1: h(c, V1170, c) = h(V1170, a, f4(f1(c)))
E2: h(V1173, V1174, V1173) = g2(V1173, V1174)
E3: h(a, V1181, f4(f1(c))) = g1(f2(f4(V1181)), f4(e))
E4: f2(f4(V1184)) = g(f3(V1184), f2(e))
E5: g(V1187, f2(e)) = g4(f1(f5(V1187)), d)
E6: f4(V1188) = V1188

- Proof :

s = g(h(c, a, c), h(a, c, a)) # step 0
 g(h(a, a, f4(f1(c))), h(a, c, a)) by E1 # step 1
 g(h(a, a, f4(f1(c))), g2(a, c)) by E2 # step 3
 g(g1(f2(f4(a)), f4(e)), g2(a, c)) by E3 # step 3
 g(g1(g(f3(a), f2(e)), f4(e)), g2(a, c)) by E4 # step 4
 g(g1(g4(f1(f5(f3(a))), d), e), g2(a, c)) by E5, E6 # step 5
```

A proof is correct (or *sound*) iff for every step  $s_i = s_{i+1}$  by  $E_{i_1}, \dots, E_{i_k}$ , the identity  $s_i = s_{i+1}$  follows logically from  $E_{i_1}, \dots, E_{i_k}$ . By Birkhoff’s seminal result (Birkhoff, Garrett 1935), this amounts to the identity  $s_i = s_{i+1}$  being derivable from the cited equations using the inference rules of equational logic.<sup>89</sup> A more algorithmic characterization of this derivability relation is based on the notion of *rewriting*. Given an equation  $e$  of the form  $l = r$ , we write  $s \rightarrow_e t$  to mean that  $s$  has a subterm at some position  $p$  such that  $sub(s, p)$  matches  $l$  wrt to a substitution  $\theta$  and  $t$  is identical to  $s$  except that  $sub(t, p) = \theta(r)$ . In words,  $t$  is obtained from  $s$  by replacing the subterm  $sub(s, p)$  that matches the left-hand side of  $e$  under  $\theta$  with the corresponding instance of the right-hand side  $r$ , i.e.,  $\theta(r)$ . We refer to  $sub(s, p)$  as a *redex* for  $e$  (or an “ $e$ -redex”) and to  $sub(t, p) = \theta(r)$  as the corresponding *contractum*. If  $s \rightarrow_e t$ , we say that  $s$  rewrites or reduces to  $t$  in one step. For a set of equations  $\mathcal{E}$ , we write  $s \rightarrow_{\mathcal{E}} t$  to mean that  $s \rightarrow_e t$  for some  $e \in \mathcal{E}$ . We write  $\leftarrow_e$  and  $\leftarrow_{\mathcal{E}}$  for the symmetric closures of  $\rightarrow_e$  and  $\rightarrow_{\mathcal{E}}$ , respectively, and  $\leftarrow_{\mathcal{E}}^*$  for the reflexive, symmetric, and transitive closure of  $\rightarrow_{\mathcal{E}}$ . As Birkhoff first noted (Birkhoff, Garrett 1935),  $s = t$  is provable from a set of equations  $\mathcal{S}$  using the inference rules of equational logic (and thus, by Birkhoff’s completeness theorem,  $s = t$  is a logical consequence of  $\mathcal{S}$ ) iff  $s \leftarrow_{\mathcal{S}}^* t$ . In our case, a proof step  $s_i = s_{i+1}$  by  $E_{i_1}, \dots, E_{i_k}$  is correct iff the identity  $s_i = s_{i+1}$  follows from the set of equations  $\mathcal{S} = \{E_{i_1}, \dots, E_{i_k}\}$ , which is to say, iff  $s_i \leftarrow_{\mathcal{S}}^* s_{i+1}$ ; and a proof as a whole is correct iff every such step is correct.

The above definition of equational proofs does not impose any requirements on the granularity of the individual steps. This has the advantage of allowing for equational proofs with widely different levels of detail, depending on one’s preferences (e.g., depending on the intended audience). On the

<sup>89</sup> There are various options for which equational inference rules to take as primitive, but a standard choice consists of reflexivity, symmetry, transitivity, and congruence (also known as substitution of equals).

flip side, it allows for proofs that are very hard—potentially impossible—to check by algorithm. In the extreme case, one could have a correct “proof” of a very complicated identity  $s = t$  with just one inference step  $s = s_0 = t$  by  $\mathcal{E}$  that cites all given axioms. To rectify this and to simplify the equational tasks considerably, we stipulate that in a step  $s_i = s_{i+1}$  by  $E_{i_1}, \dots, E_{i_k}$ , the term  $s_{i+1}$  must be obtainable from  $s_i$  by  $k$  one-step rewrites using the  $k$  cited equations, and moreover, that the relevant  $k$  redexes occur in mutually disjoint positions of  $s_i$ , and thus, in principle, all  $k$  rewrites could take place concurrently in one single step. (Two positions are disjoint iff neither is a prefix of the other.) All proofs in  $\text{EQ}_1$  adhere to this structure and have extremely simple steps that cite only one or at most two equations, which are only used for one-step reductions in the sense just described. More precisely, if  $\mathcal{S} = \{E_1, \dots, E_k\}$  is a set of  $k > 0$  equations, we write  $s \implies_{\mathcal{S}} t$  iff  $s$  contains  $k$  distinct redexes, one for each  $E_i \in \mathcal{S}$ , in  $k$  disjoint positions; and  $t$  is obtained from  $s$  by replacing each of these  $k$  redexes by the corresponding contractum. By definition, if  $s \implies_{\mathcal{S}} t$ , then  $\mathcal{S}$  does not have any redundant equations; each of the  $k$  equations in  $\mathcal{S}$  is used to reduce a unique redex in  $s$ . The reducibility relation  $\implies_{\mathcal{S}}$  is a proper subset of the usual  $\longleftarrow^*$ ; the latter is implied by the former but not conversely.

Consider the proof given in the preceding example and take the first step,  $s_0 = s_1$ :

$$g(\underline{h(c, a, c)}, h(a, c, a)) = g(\underline{h(a, a, f_4(f_1(c)))}, h(a, c, a)) \text{ by } E_1.$$

The underlined redex  $\underline{h(c, a, c)}$  of  $s_0$  matches the left-hand side of  $E_1$  under the substitution  $\theta = \{V_{1170} \mapsto a\}$ , and  $s_1$  is obtained from  $s_0$  by replacing this redex by the corresponding instance of  $E_1$ 's right-hand side, namely the contractum  $\theta(h(V_{1170}, a, f_4(f_1(c)))) = \underline{h(a, a, f_4(f_1(c)))}$ . The entire proof with underlined subterms, using red for redexes and blue for contractums, is shown below:

$$\begin{aligned} s &= g(\underline{h(c, a, c)}, h(a, c, a)) \\ &= g(\underline{h(a, a, f_4(f_1(c)))}, \underline{h(a, c, a)}) && \text{by } E_1 \\ &= g(\underline{h(a, a, f_4(f_1(c)))}, \underline{g_2(a, c)}) && \text{by } E_2 \\ &= g(\underline{g_1(f_2(f_4(a))), f_4(e)}, \underline{g_2(a, c)}) && \text{by } E_3 \\ &= g(\underline{g_1(g(f_3(a), f_2(e))), f_4(e)}, \underline{g_2(a, c)}) && \text{by } E_4 \\ &= g(\underline{g_1(g_4(f_1(f_5(f_3(a))), d), e)}, \underline{g_2(a, c)}) && \text{by } E_5, E_6 \end{aligned}$$

Here, only the penultimate term contains multiple (two) redexes; all other steps follow by replacing a single redex by a single contractum. The two redexes occur in disjoint positions, as is always the case when more than one equation is cited.

We started with 1K randomly generated examples, where each example included its own set of equational axioms and proof. All 1K proofs were correct by construction. To balance the dataset with incorrect examples, we randomly corrupted about half of the proofs by introducing errors of two types:

1. *Incorrect contractums*: On a proof step from  $s_i$  to  $s_{i+1}$ , we randomly corrupted a contractum in  $s_{i+1}$  so that  $s_{i+1}$  is no longer derivable from  $s_i$  by the equations cited by the step.
2. *Incorrect equations*: We randomly changed one of the justifying equations for a given proof step. Thus, e.g., while the original—and correct—version of the step cited equations  $E_2$  and  $E_5$ , we might change that to  $E_9$  and  $E_5$ , ensuring that the step no longer goes through.

Only one error was inserted in each corrupted proof, at a randomly chosen step.

We introduce and study three  $\text{EQ}_1$  tasks that fall along a spectrum of increasing difficulty, forming a natural progression that probes different reasoning skills:

1. *Proof checking (EQ-PC, easiest)*: This is largely a local and low-level matching and substitution task, although we will see that even here there are different gradations of difficulty depending on how much explanatory detail we demand.
2. *Equation recovery (EQ-ER, intermediate)*: This requires more search and matching than simple checking, but it's still constrained to a single step at a time.

3. *Gap filling (EQ-GF, hardest)*: This is a synthesis task. The model is given a start and end point separated by multiple elided steps and must generate a valid intermediate path. This requires multi-step lookahead, global planning, and search through a combinatorial space of possible rewrites. It's considerably more complex than the other two tasks.

### 9.3.1. Equational Proof Checking

This task contains 200 randomly chosen examples from the set of 1K equational proofs. The slice is balanced so that about half of the chosen proofs are correct, while the others are corrupted as described above. The model is asked to determine whether the proof is correct, and more importantly, to *justify* its answer. Accordingly, the requested output has two components: a binary correctness verdict and an explanation/justification for the verdict.

Checking an equational proof in detail means verifying that each correct step goes through owing to the cited identities, and not only identifying incorrect steps but pinpointing the exact source of error. In particular, if a model claims that a proof with  $n$  steps is correct, then its full explanation must be a sequence of  $n$  structured objects, one for each proof step

$$s_i = s_{i+1} \text{ by } E_{i_1}, \dots, E_{i_k}$$

for  $i = 0, \dots, n - 1$ , elaborating *why* each step is correct. The structured object consists of the relevant proof step index (the number  $i + 1$ ) along with a list of rewrites corresponding to the  $k$  redexes in  $s_i$ . Each rewrite is a dictionary with the following keys:

- *redex*: the redex in question;
- *position*: the exact position of the redex in  $s_i$ ;
- *equation*: the identity whose left-hand side matches the redex;
- *contractum*: the corresponding contractum in  $s_{i+1}$ .

More concretely, the model's output must be a dictionary with two keys, *correct* and *explanation*. The value of *correct* is *true* or *false*, depending on whether the model determines that the input proof is correct. When *correct* is *true*, the value of *explanation* must be a list of dictionaries, each with two keys, *step* (the step number, one for each step of the proof) and *rewrites*, a list of rewrites of the form described above. Thus, for the example in the preceding section, an appropriate output would be:

```
{ "correct": true,
 "explanation":
 [{ "step": 1,
 "rewrites": [{ "redex": "h(c,a,c)", "position": [1],
 "equation": "E1", "contractum": "h(a,a,f4(f1(c)))" }] },
 { "step": 2,
 "rewrites": [{ "redex": "h(a,c,a)", "position": [2],
 "equation": "E2", "contractum": "g2(a,c)" }] },
 { "step": 3,
 "rewrites": [{ "redex": "h(a,a,f4(f1(c)))", "position": [1],
 "equation": "E3", "contractum": "g1(f2(f4(a)),f4(e))" }] },
 { "step": 4,
 "rewrites": [{ "redex": "f2(f4(a))", "position": [1,1],
 "equation": "E4", "contractum": "g(f3(a),f2(e))" }] },
 { "step": 5,
 "rewrites": [{ "redex": "g(f3(a),f2(e))", "position": [1,1],
 "equation": "E5", "contractum": "g4(f1(f5(f3(a))),d)",
 { "redex": "f4(e)", "position": [1,2],
 "equation": "E6", "contractum": "e" } }] }] }
```

By contrast, if *correct* is *false*, then the value of a complete *explanation* must be a single dictionary corresponding to the first incorrect step in the proof, and should be:

1. either of the form

```
{ "step": ...,
 "equation": ...,
 "position": ...,
 "expectedContractum": ...,
 "actualContractum": ... }
```

indicating that the contractum for the equation and redex in question (at the given position) is not what was expected by the cited equation;

2. or of the form

```
{ "step": ...,
 "givenEquations": [...],
 "correctEquations": [...],
```

indicating that the cited identities do not succeed in justifying that step, though the identities listed under *correctEquations* do.

Outputs of the first form capture errors introduced by corrupting contractums; outputs of the second form capture errors introduced by noising cited identities.

As an example of an incorrect proof and the proper corresponding output, consider the following input:

```
- Axioms:
E1: h(a, V1110, V1111) = g3(f(V1111), f1(V1110))
E2: h(b, c, V1118) = g(f4(f3(V1118)), a)
E3: g3(f(c), f1(V1119)) = g(f2(V1119), g1(a, c))
E4: f2(V1124) = f1(f1(V1124))
E5: f4(f3(V1125)) = f2(g(V1125, f4(c)))

- Proof:
s = r(h(e, e, c), g(c, b), h(b, c, b), h(a, a, c))
 r(h(e, e, c), g(c, b), h(b, c, b), g3(f(c), f1(a))) by E1 # step 0
 r(h(e, e, c), g(c, b), g(f4(f3(b)), a), g(f2(a), g1(a, c))) by E2, E3 # step 1
 r(h(e, e, c), g(c, b), g(f2(g(b, f4(e))), a), g(f2(a), g1(a, c))) by E4, E5 # step 2
 r(h(e, e, c), g(c, b), g(f2(g(b, f4(e))), a), g(f2(a), g1(a, c))) by E4, E5 # step 3
```

Here, the proof proceeds correctly until the last step:

$$\begin{aligned} & \vdots \quad \vdots \quad \vdots \\ & = r(h(e, e, c), g(c, b), g(\underline{f_4(f_3(b))}, a), g(f_2(a), g_1(a, c))) \quad \text{by } E_2, E_3 \\ & = r(h(e, e, c), g(c, b), g(\underline{f_2(g(b, f_4(e)))}, a), g(f_2(a), g_1(a, c))) \quad \text{by } E_4, E_5 \end{aligned}$$

The only redex in the penultimate term that matches the left-hand side of  $E_5$  is in position [3,1], shown underlined in red:  $f_4(f_3(b))$ . The correct contractum for this redex is  $f_2(g(b, f_4(c)))$ , but the one that actually appears in the last term is  $f_2(g(b, f_4(e)))$ . Accordingly, an appropriate output for this proof is:

```
{ "correct": false,
 "explanation": { "step": 3,
 "equation": "E5",
 "position": [3, 1],
 "expectedContractum": "f2(g(b, f4(c)))",
 "actualContractum": "f2(g(b, f4(e)))" }
```

Finally, here is an example illustrating the second type of error, the citing of incorrect equations:

```
- Axioms:
E1: g(V1068, d) = f(f2(f3(V1068)))
E2: g(b, V1071) = h(V1071, f4(a), d)
E3: g(b, V1072) = f5(V1072)
E4: f2(V1073) = f1(V1073)
E5: f3(V1080) = V1080
E6: f4(V1083) = g2(V1083, f5(b))
E7: f(V1086) = f1(V1086)

- Proof:
s = h(g(b, d), g(b, b), g(b, e))
 h(f(f2(f3(b))), g(b, b), g(b, e)) by E1 # step 0
 h(f(f1(f3(b))), f5(b), h(e, f4(a), d)) by E2, E3, E4 # step 1
 h(f(f1(b)), f5(b), h(e, f4(a), d)) by E5 # step 2
 h(f(f1(b)), f5(b), h(e, g2(a, f5(b)), d)) by E2 # step 3
 h(f1(f1(d)), f5(b), h(e, g2(a, f5(b)), d)) by E7 # step 4
 h(f1(f1(d)), f5(b), h(e, g2(a, f5(b)), d)) by E7 # step 5
```

The error here is on step 4:  $E_2$  does not justify the passage from  $h(f(f_1(b)), f_5(b), h(e, f_4(a), d))$  to

$$h(f(f_1(b)), f_5(b), h(e, g_2(a, f_5(b)), d)).$$

There are no redexes in  $h(f(f_1(b)), f_5(b), h(e, f_4(a), d))$  that match the left-hand side of  $E_2$ . However, the step is justified by  $E_6$ , which rewrites the redex  $f_4(a)$  to  $g_2(a, f_5(b))$ . Thus, the correct output in this case is:

```
{ "correct": false,
 "explanation": { "step": 4,
 "givenEquations": ["E2"],
 "correctEquations": ["E6"]}}
```

Our few-shot prompt for this task contains several ICL examples of both correct and incorrect proofs, along with appropriate corresponding outputs.

What was described above is a task that requires thorough and detailed checking of equational proofs. We refer to this as *level-3 proof checking*. This is what our proof-checking prompt for equational logic specifies. Nevertheless, to paint a richer performance picture, we distinguish two additional evaluation modes that require less detail. In total, we have:

- *Level 1*: The model is given full credit as long as it can guess whether the proof is correct, i.e., as long as it makes the right binary judgment. Accuracy at this level measures a model’s ability to get the “vibe” of the proof. Is it roughly correct or does something seem off?
- *Level 2*: If the model rightly claims that the proof is correct, then it must provide detailed evidence for its claim, i.e., the *explanation* it outputs in that case must be fully correct. This level makes no demands on the model’s output for incorrect proofs. As long as the model correctly guesses that the proof has an error, it is given full credit even if it gets the error details wrong. Thus, level-2 accuracy measures a model’s ability to perform the computational work of positive verification: identifying redexes, matching them to equations, and confirming the contractums.
- *Level 3*: As already mentioned, level-3 proof checking is the most thorough: credit is given only if the verdict is correct and the model’s *explanation* gets all the details right, both for correct and for incorrect proofs. Thus, this level requires everything required by the previous levels, as well as debugging skills, or the ability to perform error analysis by precisely identifying and explaining the first point of failure in an incorrect proof. This is the highest level of proof-checking scrutiny. The model must keep track of the “expected” state (what the term should look like after the rewrite); compare it against the actual state; and then articulate the discrepancy. This level also requires a degree of error-repair ability, insofar as the model must single out the correct (or “expected”) contractum in the case of a contractum corruption, or the correct equations in the case of an equation corruption.

These levels form a ladder of proof-checking rigor: proof-checking success at L3 (level 3) entails L2 success, which entails L1 success, but the converses do not hold.

Model	L1	L2	L3	Model	L1	L2	L3
gemini-3.1-pro-preview	0.98 (±0.02)	0.93 (±0.04)	<b>0.93</b> (±0.04)	gpt-5.1	0.90 (±0.04)	0.81 (±0.05)	0.76 (±0.06)
gpt-5.2	0.97 (±0.03)	0.82 (±0.05)	0.75 (±0.06)	gemini-3.0-pro-preview	0.98 (±0.02)	0.76 (±0.06)	0.74 (±0.06)
claude-opus-4-5-20251101	0.99 (±0.02)	0.74 (±0.06)	0.71 (±0.06)	gpt-5	0.95 (±0.03)	0.77 (±0.06)	0.69 (±0.06)
o3	0.88 (±0.05)	0.69 (±0.06)	0.58 (±0.07)	grok-4	0.90 (±0.04)	0.67 (±0.06)	0.56 (±0.07)
claude-sonnet-4.5-ext-thinking	0.97 (±0.03)	0.64 (±0.07)	0.56 (±0.07)	deepseek-reasoner	0.89 (±0.04)	0.62 (±0.07)	0.53 (±0.07)
gemini-2.5-pro	0.87 (±0.05)	0.56 (±0.07)	0.52 (±0.07)	o4-mini	0.69 (±0.06)	0.62 (±0.07)	0.43 (±0.07)
o3-mini	0.67 (±0.06)	0.52 (±0.07)	0.25 (±0.06)	deepseek-r1-0528	0.72 (±0.06)	0.51 (±0.07)	0.25 (±0.06)
claude-opus-4-20250514	0.74 (±0.06)	0.36 (±0.07)	0.25 (±0.06)	deepseek-v3.1	0.64 (±0.07)	0.52 (±0.07)	0.24 (±0.06)
claude-sonnet-4-20250514	0.61 (±0.07)	0.53 (±0.07)	0.23 (±0.06)	grok-3	0.64 (±0.07)	0.38 (±0.07)	0.14 (±0.05)
hermes-4	0.58 (±0.07)	0.50 (±0.07)	0.08 (±0.04)	deepseek-v3	0.49 (±0.07)	0.49 (±0.07)	0.06 (±0.03)
gpt-4o	0.62 (±0.07)	0.32 (±0.06)	0.04 (±0.03)	llama-3.1-405b	0.53 (±0.07)	0.06 (±0.03)	0.04 (±0.03)
gpt-4.1	0.52 (±0.07)	0.06 (±0.03)	0.02 (±0.02)	mistral-magistral	0.53 (±0.07)	0.46 (±0.07)	0.01 (±0.02)

**Table 28.** Equational proof-checking accuracies at 3 levels of detail.

Results for all three levels are shown in Table 28. For L1, the random-guessing accuracy baseline is already 50%. Thus, above-chance accuracy for the top-performing model, Gemini 3.1, is 98%-50%

= 48%. If we normalize, we can say that Gemini’s accuracy is  $(0.98 - 0.5)/(1 - 0.5) = 0.96$ , i.e., that Gemini 3.1 is achieving 96% of the possible improvement over random guessing. This is near-perfect performance @L1. However, beyond the top 13 models, performance drops to below 70% even at L1. So top-tier and some mid-tier models have very strong “intuition” that can reliably distinguish correct equational proofs from broken ones, but the rest have a lot of room for improvement. The worst-performing models do about as well as random guessing.

Performance across the board degrades sharply when we move to L2 and require models to not simply guess correctness but explain on a step-by-step basis *why* the proof is correct, namely, which are the relevant redexes matching the cited equations, where exactly they are located in the preceding step’s term, and what are the corresponding contractums. Opus 4.5 experiences a 25-absolute-point drop in going from L1 to L2, while for Gemini 3 and GPT-5.2 the drop is 22 and 15 absolute points, respectively. This shows that even for elite models, there is a large difference between simply guessing that a proof is correct and properly articulating its correctness in a structured explanation. Gemini 3.1 is the only top model whose decline is relatively contained (5 points only).

Performance drops further at L3, which extends the detailed-explanation requirement to incorrect proofs. At this level, a model must do everything it must do for L2 but also document where and how an incorrect proof fails. For some top models (especially for Gemini 3.1 and Gemini 3), the L2 → L3 drop is basically zero, and considerably smaller than their L1 → L2 drops. This suggests that these models are competent debuggers, almost as good at diagnosing errors as they are at verifying correctness. By contrast, most mid-tier models (like DeepSeek-R1-0528, o4-mini, and o3-mini), experience catastrophic L2 → L3 drops. These models are much better at verifying a correct stretch of equational reasoning than diagnosing a flawed one.

Taking the L3 numbers as the main set of results, the DI of this task is an exceptionally high 0.33.

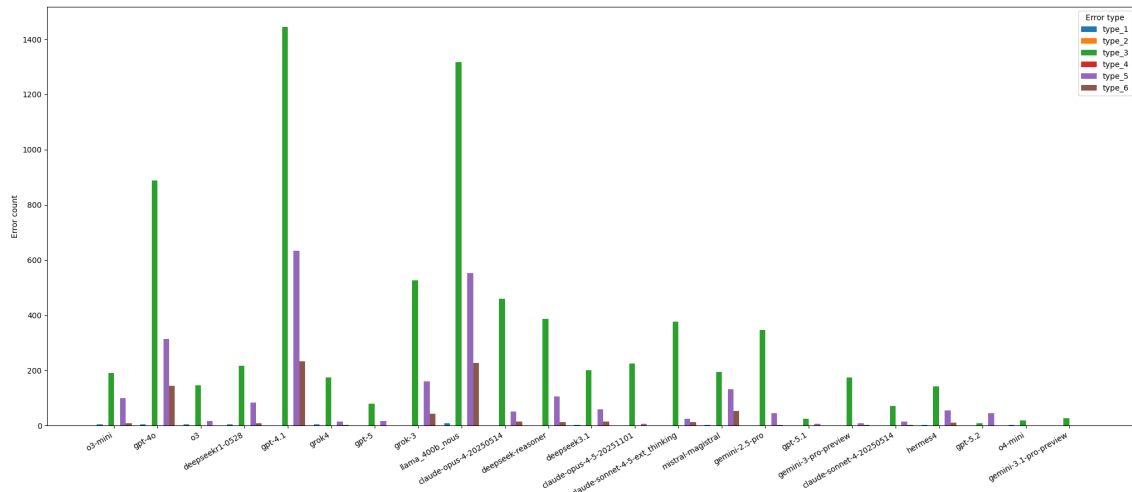
The L2 error distribution is shown in Fig. 40. We distinguish 6 error types at this level. Recall that, at L2, if a model declares a proof correct then its explanation must include a list of rewrites for every proof step, where every rewrite is a dictionary with four keys: *position*, *redex*, *contractum*, and *equation*. Error types 3–6 correspond precisely to one of these 4 values being wrong.

1. *Type 1*: Ill-formatted responses, such as a JSON-schema violation (e.g., a response dictionary that does not include one of the expected keys) or an explanation for a proof that is declared correct but is missing some of the proof steps.<sup>90</sup>
2. *Type 2*: Wrong correctness verdict.
3. *Type 3*: Wrong position in a rewrite of an explanation step.
4. *Type 4*: Wrong redex in a rewrite of an explanation step.
5. *Type 5*: Wrong contractum in a rewrite of an explanation step.
6. *Type 6*: Wrong equation in a rewrite of an explanation step.

Error types 1 and 2 are mutually exclusive with types 3–6, but the latter can co-occur in an explanation (and even on a single explanation step), so Fig. 40 reports raw error counts summed over all proof steps. For that reason, the absolute values of these counts cannot be meaningfully compared across models with widely different accuracies on the set of correct proofs. For example, claude-sonnet-420250514 makes far fewer errors than gemini-2.5-pro, but that’s because it only recognizes 27 out of the 103 correct proofs, whereas gemini-2.5-pro correctly recognizes 91, so naturally it will have more opportunities to make errors of types 3–6.

The plot shows that misidentifying redex positions (type-3 errors) is the most common failure mode at L2. Models usually output positions that are close to the correct position but do not actually match it. We quantify this closeness relation by defining a similarity metric that considers how much of the path is shared between the correct position and the model-generated position before the two diverge, i.e., their longest common prefix. The more steps they share from the root, the

<sup>90</sup> We are lenient in these judgments, e.g., instead of checking for strict equality between the set of expected dictionary keys and the set of generated keys, we check only for a subset relation, as some models unilaterally insert extra key-value entries, such as a *note* that provides additional commentary about something.



**Figure 40.** Level-2 error distribution for equational proof checking.

more similar they are. The metric is normalized so the score always lies between 0 and 1. If the two positions are identical, the score is 1; if they diverge immediately at the root, the score is 0. For intermediate cases, it strikes a balance between the amount of common structure and the total lengths of the paths. For instance,  $[3, 3, 1]$  and  $[3, 3]$  share a prefix of length 2 and one is a prefix of the other, so their similarity is high (0.8), while paths that diverge early get much lower scores. The mean similarities for all models across their type-3 errors are as follows: gemini-3.1-pro-preview: 0.83, gpt-5.2: 0.77, gpt-5.1: 0.75, gpt-5: 0.76, gemini-3.0-pro-preview: 0.75, o3: 0.71, o4-mini: 0.70, claude-sonnet-4-20250514: 0.69, claude-opus-4-5-20251101: 0.68, grok-4: 0.66, gemini-2.5-pro: 0.66, deepseek-r1-0528: 0.65, deepseek-reasoner: 0.65, claude-sonnet-4.5-ext-thinking: 0.65, grok-3: 0.62, opus-4: 0.61, o3-mini: 0.60, deepseek-v3.1: 0.53, gpt-4o: 0.5, gpt-4.1: 0.5, llama-3.1-405b: 0.45, hermes-4: 0.41, mistral-magistral: 0.38. (Note: we abbreviated claude-opus-4-20250514 as opus-4.) DeepSeek-V3 makes no type-3 errors. These numbers track only modestly with overall performance, with some exceptions such as claude-sonnet-4-20250514, which is better at locating precise positions than both gemini-2.5-pro and grok-4, suggesting that this might be a fairly distinct skill, not just a corollary of general reasoning ability.

The results indicate that most models are fairly good at “what” and “why” but bad at “where.” They can often correctly identify the string being changed (redex), what it changes into (contractum), and even the rule that governs the change (equation); but they fail at precisely locating that change within the larger tree structure. This should not be surprising, given that models do not explicitly build or traverse syntax trees. Tree structure is inherently two-dimensional, so a transformer-based model must infer it from syntactic cues in the 1D token sequence that it actually sees, primarily from parentheses and commas, which tends to be fuzzy and imprecise for the purposes of exact hierarchical localization.

GPT-5.2 stands out here as having the strongest localization skills. As mentioned earlier, raw counts of type-3 errors cannot be compared across models that accept widely different numbers of correct proofs. However, for models that have similar acceptance rates on correct proofs, comparing the counts of error types 3—6 can reveal strikingly different failure regimes. Focusing specifically on type-3 errors, we see that GPT-5.2 accepts 98 out of the 103 correct proofs and makes a total of only 8 type-3 errors. By contrast, Gemini 3 Pro Preview accepts 102 out of the 103 correct proofs and makes 175 such errors. Normalizing, we get a type-3 error rate or 0.08 for GPT-5.2 and 1.7 for Gemini 3, a difference of about 1.32 orders of magnitude. Even Gemini 3.1, the top-performing model, makes 26 type-3 errors, for an error rate of 0.25, about 3 times larger than GPT-5.2.

We also separately compute the number of completely invalid positions that models generate (i.e., any position for a term  $t$  that is not a member of  $Dom(t)$ ), summed over all explanation steps. To

normalize, we again divide each count by the number of correct proofs recognized by the corresponding model, which yields the following mean number of invalid positions generated per correct proof (DeepSeek-V3 is again excluded because it only recognizes 2 correct proofs):

gpt-4.1	6.94	hermes-4	3.25	claude-sonnet-4-20250514	0.85
gpt-4o	5.85	deepseek-v3.1	3.14	gemini-3.0-pro-preview	0.74
llama-3.1-405b	5.08	claude-opus-4-20250514	2.36	o3	0.70
mistral-magistral	4.27	deepseek-reasoner	2.12	gpt-5	0.33
grok-3	3.29	gemini-2.5-pro	2.05	o4-mini	0.12
o3-mini	1.27	claude-sonnet-4.5-ext-thinking	1.95	gpt-5.1	0.09
deepseek-r1-0528	1.25	grok-4	0.95	gemini-3.1-pro-preview	0.06
claude-opus-4-5-20251101	0.92	gpt-5.2	0.02		

How well models identify the precise positions of subterms of interest, such as redexes, is likely indicative of their general ability to navigate deeply nested tree-structured data.

Among exclusively L3 errors, the most common type for most models is misidentifying the step where the proof first goes wrong. Further, from the two types of possible proof errors, one due to contractum corruption and one due to corruption of the justifying equations, models have a harder time pinpointing the details of the former. They struggle roughly equally in identifying the expected contractum and the actual contractum. The discrepancy makes sense. An equation corruption is an easier diagnosis: “I see a change from  $u$  to  $v$ . The cited rule is  $E_i$ , but this rule doesn’t apply at all.” A contractum corruption, by contrast, is much harder: “I see a change from  $u$  to  $v$ . The cited rule is  $E_i$ . The redex  $u$  matches the LHS of  $E_i$ , so the rule does apply. Now, I must compute the expected contractum  $v'$  and compare it to the actual  $v$ .” This requires the model to perform a forward inference step itself, hold the result in memory, and then perform a comparison, so it’s more computationally demanding.

### 9.3.2. Recovery of Elided Equations

This task also contains 200 samples drawn from the set of 1K examples, evenly split between correct and incorrect proofs. The input given to the model is the same as in the previous task, a set of named equations  $\mathcal{E}$  and a proof  $D$ , except that all justifying equations are now omitted from every step in  $D$  and the model has to discover them on its own. Specifically, for every step of the proof that derives a term  $s_{i+1}$  from a term  $s_i$ , the model has to output a set  $\mathcal{S}_i \subseteq \mathcal{E}$  of equations that justify the step in the sense that  $s_i \implies_{\mathcal{S}_i} s_{i+1}$ , assuming that such a set  $\mathcal{S}_i$  exists. If it does not, e.g., because the step from  $s_i$  to  $s_{i+1}$  has an error, then the correct output is the empty set of equations. For example, given the input:

```
- Equations:

E1: h(c, V1170, c) = h(V1170, a, f4(f1(c)))
E2: h(V1173, V1174, V1173) = g2(V1173, V1174)
E3: h(a, V1181, f4(f1(c))) = g1(f2(f4(V1181)), f4(e))
E4: f2(f4(V1184)) = g(f3(V1184), f2(e))
E5: g(V1187, f2(e)) = g4(f1(f5(V1187)), d)
E6: f4(V1188) = V1188

- Proof:

s = g(h(c, a, c), h(a, c, a)) # step 0
 g(h(a, a, f4(f1(c))), h(a, c, a)) # step 1
 g(h(a, a, f4(f1(c))), g2(a, c)) # step 2
 g(g1(f2(f4(a)), f4(e)), g2(a, c)) # step 3
 g(g1(g(f3(a), f2(e)), f4(e)), g2(a, c)) # step 4
 g(g1(g4(f1(f5(f3(a))), d), e), g2(a, c)) # step 5
```

the following list would be a correct output:

```
[{"step": 1, "supportingEquations": ["E1"]},
 {"step": 2, "supportingEquations": ["E2"]},
 {"step": 3, "supportingEquations": ["E3"]},
 {"step": 4, "supportingEquations": ["E4"]},
 {"step": 5, "supportingEquations": ["E5", "E6"]}]
```

Note that this task has no unique correct output for a given step from  $s_i$  to  $s_{i+1}$ , as there may be multiple sets of equations  $\mathcal{S} \subseteq \mathcal{E}$  such that  $s_i \implies_{\mathcal{S}} s_{i+1}$  (although typically the number of possibilities is small). We thus verify each answer independently. If an answer  $\mathcal{S}$  is a non-empty set of equations, we check that  $s_i \implies_{\mathcal{S}} s_{i+1}$ . If an answer is the empty set of equations, we check that there is no  $\mathcal{S} \subseteq \mathcal{E}$  such that  $s_i \implies_{\mathcal{S}} s_{i+1}$ . This is easily decidable, first by filtering those candidate equations that match some subterm of  $s_i$ , and then checking whether every disagreement between  $s_i$  and  $s_{i+1}$  can be resolved by one of these candidates.<sup>91</sup>

This task has a strong abductive-verification flavor. For every step of the proof, the model is faced with three pieces of “evidence”: the starting state  $s_i$ , the ending state  $s_{i+1}$ , and the equational axioms in  $\mathcal{E}$  as the permissible “physical laws.” The claim to be investigated is that  $s_{i+1}$  is easily derivable from  $s_i$  by a set of equations  $\mathcal{S} \subseteq \mathcal{E}$  (where easy derivability means no deduction chains, just single-step reductions as defined by  $\implies_{\mathcal{S}}$ ). To decide the claim, the model must work backward from the evidence by looking at the “effect” (the difference between  $s_i$  and  $s_{i+1}$ ) and trying to find a “cause” (a minimal set of axioms) that would explain it. This can be performed by difference analysis followed by hypothesis generation. Difference analysis aims to identify the subterm(s) that changed in the transition from  $s_i$  to  $s_{i+1}$ ; this is the locus of the analysis. Each subterm change consists of a redex candidate in  $s_i$  and a transformed version in  $s_{i+1}$ , for which the model generates hypotheses: “Could axiom  $E_1$  explain this transformation? Could  $E_2$ ? Could  $E_3$ ? . . .”

While the task requires processing a good deal of highly structured data, it lends itself to pattern recognition and does not require fine-grained analysis. In the running example, for instance, the first step can only be justified by  $E_1$ , since that’s the only equation whose right-hand side can introduce a subterm of the form  $f_4(f_1(c))$ . Likewise, the second step is easily seen to follow by  $E_2$ .

We report two metrics on this task. The main one is *proof-level accuracy* (or PLA for short): the fraction of proofs for which the model generates correct outputs at every step. The secondary metric is *step-level accuracy* (SLA): the proportion of individual proof steps for which the model produces correct outputs. The former is clearly more demanding, as a model is rewarded only if it handles correctly *every* step of the proof. The step-level metric is much more forgiving, as it rewards partial success on individual steps. Proof-level and step-level accuracies are shown in Table 29.

Model	PLA	SLA	Model	PLA	SLA
gpt-5.2	0.94 (± 0.03)	0.99 (± 0.02)	gpt-5.1	0.92 (± 0.04)	0.99 (± 0.02)
gemini-3.1-pro-preview	0.79 (± 0.06)	0.98 (± 0.02)	gemini-2.5-pro	0.78 (± 0.06)	0.98 (± 0.02)
gpt-5	0.77 (± 0.06)	0.98 (± 0.02)	claude-opus-4-5-20251101	0.70 (± 0.06)	0.97 (± 0.03)
gemini-3.0-pro-preview	0.69 (± 0.06)	0.97 (± 0.03)	o3	0.67 (± 0.06)	0.96 (± 0.03)
claude-sonnet-4.5-ext-thinking	0.62 (± 0.07)	0.96 (± 0.03)	grok-4	0.60 (± 0.07)	0.94 (± 0.03)
deepseek-reasoner	0.44 (± 0.07)	0.54 (± 0.07)	o3-mini	0.39 (± 0.07)	0.83 (± 0.05)
deepseek-r1-0528	0.39 (± 0.07)	0.81 (± 0.05)	deepseek-v3.1	0.35 (± 0.07)	0.79 (± 0.06)
claude-opus-4-20250514	0.33 (± 0.06)	0.90 (± 0.04)	o4-mini	0.29 (± 0.06)	0.80 (± 0.06)
claude-sonnet-4-20250514	0.29 (± 0.06)	0.85 (± 0.05)	hermes-4	0.17 (± 0.05)	0.57 (± 0.07)
grok-3	0.12 (± 0.05)	0.74 (± 0.06)	deepseek-v3	0.09 (± 0.04)	0.72 (± 0.06)
gpt-4.1	0.07 (± 0.04)	0.46 (± 0.07)	mistral-magistral	0.02 (± 0.02)	0.30 (± 0.06)
llama-3.1-405b	0.01 (± 0.02)	0.16 (± 0.05)	gpt-4o	0.00 (± 0.01)	0.04 (± 0.03)

**Table 29.** EQ-ER performance; PLA = Proof-Level Accuracy, SLA = Step-Level Accuracy.

The SLA metric gauges raw ER capability. Top models have basically perfect internal mechanisms for solving this problem. PLA, by contrast, measures reliability and consistency. Here only two models manage to score higher than 80%: GPT-5.2 and GPT-5.1, with the former getting a nearly perfect score. The rest of the models, even elite models like Gemini 3, are not reliable enough to get every step right consistently across an entire multi-step proof. A single mistake in a 20-step proof is enough to fail the PLA metric. This shows that moving from high capability to high reliability in sequential reasoning tasks remains a critical frontier for most LLMs.

<sup>91</sup> The set  $D$  of “disagreements” between two terms  $s$  and  $t$  consists of those minimal positions  $p \in \text{Dom}(s) \cap \text{Dom}(t)$  such that the subterms  $\text{sub}(s, p)$  and  $\text{sub}(t, p)$  have different roots. By “minimal” we mean that if  $p \in D$  and  $q$  is a strict suffix of  $p$  then  $q \notin D$ . Thus, this set is simply the set of earliest or highest positions where the two trees have a mismatch.

PLA's DI (discriminability index) is exceptionally high at 0.36.

### 9.3.3. Error Analysis

We partition the set of all possible step-level errors into 4 types:

1. The model's output is ill-formed.
2. The model gives the empty list of supporting equations for a step from  $s_i$  to  $s_{i+1}$ , indicating that the step cannot be  $\implies_S$ -justified by any  $S \subseteq \mathcal{E}$ , when in fact it can.
3. The model gives a non-empty list of supporting equations  $S$  for a step when in fact the step cannot be  $\implies_S$ -justified by any  $S \subseteq \mathcal{E}$ .
4. The model gives a non-empty list of supporting equations  $S$  for a step and the step *is* justifiable, but the particular equations given in  $S$  do not justify the step.

A proof-level error is this: Every element of the list given by the model is properly formatted, but instead of containing an element for every step of the proof, the list is cut off prematurely or only contains answers for some of the proof steps, but not all of them. We refer to this as a type-5 error.

Step-level rates for the 4 possible error types are shown in Figure 41. Note that the SLAs of

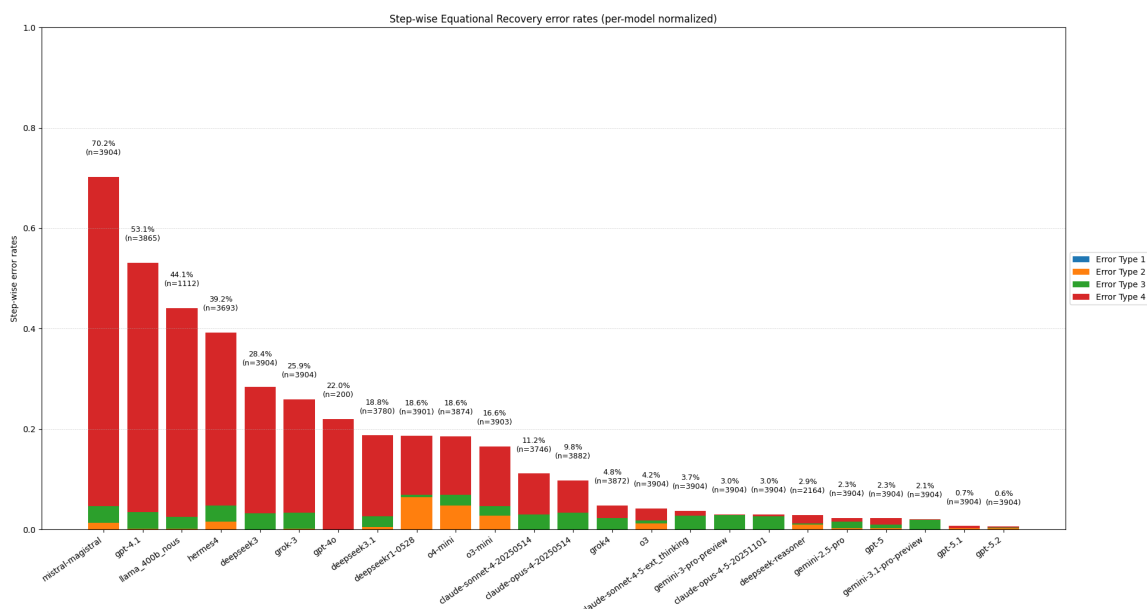


Figure 41. Normalized distribution of step-level ER errors.

Table 29 are not equal to 1 minus these error rates. That is because the error rates of Fig. 41 are normalized with respect to the total number of steps that the model has attempted to explain (these are the values  $n = \dots$  shown above each bar in the figure). By contrast, the SLAs of Table 29 are normalized with respect to the total number of all steps across all proofs, regardless of whether the model has provided a response for a step. As an example, consider gpt-4o and llama-3.1-405b. These models only provided answers for a small fraction of all steps, particularly gpt-4o, which only handled 200 out of a total of 3904 steps, a mere 5%. In fact, only 10 models engaged with all possible steps; the rest omitted at least a few. The number of steps omitted (the difference between 3904 and the value of  $n$  shown in the figure) is a direct absolute measure of a model's type-5 errors.

No model makes a formatting error. The large majority of ER errors are of type 4. Models correctly identify that the step is valid but they provide the wrong set of justifying equations. This is not a failure of pattern recognition in the simple sense. The model is usually "in the right ballpark" but lacks precision. It suggests a failure of meticulous verification. Models seem to rely on heuristics like "Axiom  $E_5$  looks very similar to the change I see" and then confidently output  $E_5$  without performing the rigorous computational step of applying the substitution and ensuring that it produces exactly the target term.

Type-2 errors (where the model misclassifies the step as unjustifiable by outputting  $\square$ ) are a different failure mode that cannot be explained simply by lower reasoning competence, since there are several models with poor performance but no type-2 errors. The rate of type-2 errors likely depends on whether a model's post-training alignment prioritizes helpfulness and engagement versus abstention in the presence of uncertainty. Competence still determines how often the model is uncertain, but whether the model tries to engage despite the uncertainty is largely shaped by its alignment regimen.

Another finding involves bidirectional rewriting. Our formalization explicitly defines  $\implies_S$  as left-to-right only, which suffices for this dataset. Yet several models repeatedly choose sets of equations that can only justify a step if reduction is performed from right to left. Strictly speaking, these answers are incorrect because they do not adhere to the prompt instructions, which only allow for left-to-right rewriting. We give full credit to such answers anyway, because they are logically correct, even if not consistent with the prompt. While this happens relative rarely,<sup>92</sup> it is curious that a model would choose equations that require right-to-left rewriting when equally simple equations that work from left to right (and are thus fully consistent with the prompt) are available. This suggests that many models are following the spirit of equational reasoning (proving two things are the same) while violating the letter of the specific task instructions (only rewriting from left to right). In other words, their deeply ingrained semantic understanding of equality as a symmetric relation is overriding the specific procedural constraints of the task. This underscores how deeply learned priors from pretraining (such as the symmetry of equality) impact a model's operation and points to an alignment challenge in getting models to follow instructions that might feel "unnatural."

### 9.3.4. Equational Gap Filling

This task is also a subset of EQ<sub>1</sub> with 200 proofs, but each proof has a gap of a randomly chosen size inserted at a randomly chosen location. The model is asked to fill that gap with an appropriate sequence of steps. The output must be a dictionary with two main keys, *explanation* and *missingSteps*. The value of *explanation* is a plain text string describing the model's reasoning strategy. The value of *missingSteps* is the main payload: the subproof filling the gap. This is expressed as a list of dictionaries representing proof steps, each with two keys, *term* and *supportingEquations*. Thus, such a list can be viewed as a sequence of pairs  $(t_i, N_i)$ ,  $1 \leq i \leq k$  for some  $k \geq 1$ , where each  $t_i$  is a term (the value of the key *term*) and  $N_i$  is a list of equation names (the value of the key *supportingEquations*). The specified steps correctly fill the gap iff the proof

$$\begin{aligned} s' &= t_1 \text{ by } N_1 \\ &= t_2 \text{ by } N_2 \\ &\vdots \\ &= t_k \text{ by } N_k \\ &= t' \text{ by } \dots \end{aligned} \tag{15}$$

correctly derives the equation  $s' = t'$ , where  $s'$  is the term that appears on the step immediately before the gap and  $t' \text{ by } \dots$  is the step immediately after the gap. More precisely, we must have  $s' \iff_{N_1} t_1$ ;  $t_k \iff \dots t'$ ; and  $t_{i-1} \iff_{N_i} t_i$  for every  $i = 2, \dots, k$ , where  $\iff_S$  is the symmetric closure of  $\implies_S$ . (Thus, unlike ER, this task explicitly allows for bidirectional rewriting.) If the model determines that there is no way to fill the gap, it must output the empty list as the value of *missingSteps*. We write ?? to indicate the presence of a gap. We explicitly require  $|N_i| \leq 2$ , i.e., to avoid trivializing the problem, a model may not cite more than 2 equations per step.

For example, given the input:

```
- Equations:
```

<sup>92</sup> o3-mini leads the pack, with bidirectional rewrites in 32 proofs (out of the total 200); followed by GPT-4.1 (17 proofs). Many others use bidirectional rewriting more sporadically.

```

E1: h(c,V1170,c) = h(V1170,a,f4(f1(c)))
E2: h(V1173,V1174,V1173) = g2(V1173,V1174)
E3: h(a,V1181,f4(f1(c))) = g1(f2(f4(V1181)),f4(e))
E4: f2(f4(V1184)) = g(f3(V1184),f2(e))
E5: g(V1187,f2(e)) = g4(f1(f5(V1187)),d)
E6: f4(V1188) = V1188

```

- Proof:

```

s = g(h(c,a,c),h(a,c,a))
 g(h(a,a,f4(f1(c))),h(a,c,a)) by E1
 g(h(a,a,f4(f1(c))),g2(a,c)) by E2
 ?? ??
 g(g1(g(f3(a),f2(e)),f4(e)),g2(a,c)) by E4
 g(g1(g4(f1(f5(f3(a))),d),e),g2(a,c)) by E5, E6

```

a correct output would be:

```

{"missingSteps": [{"term": "g(g1(f2(f4(a)),f4(e)),g2(a,c))",
 "supportingEquations": ["E3"]}]}

```

since this one step is sufficient to fill the gap. As a simple example of a gap that cannot be filled, consider:

- Equations:

```

E1: h(V1130,V1131,V1132) = h4(V1130,f(V1132),f2(V1131))
E2: h(c,c,V1149) = g3(V1149,c)
E3: g(V1152,V1153) = f2(V1152)
E4: h4(e,f(V1154),f2(a)) = g3(f2(f1(V1154)),e)
E5: r(d,c,V1162,V1163) = g(g3(V1163,V1162),f(f4(e)))

```

- Proof:

```

s = r(r(d,c,d,b),h(c,c,c),g(d,c),h(e,a,c))
 r(r(d,c,d,b),h(c,c,c),g(d,c),h4(e,f(c),f2(a))) by E1
 r(r(d,c,d,b),g3(c,c),g(d,c),h4(e,f(c),f2(a))) by E2
 ?? ??
 r2(g(g3(b,d),f(f4(e))),g3(c,c),f2(d),g3(f2(f1(c)),e)) by E5

```

Here the correct value for *missingSteps* would be the empty list [], because there are no equations that can introduce an occurrence of  $r_2$ . Only a small minority of examples (7 out of 200) have gaps that cannot be filled.

In addition to *explanation* and *missingSteps*, the top-level JSON output for this task must contain a third key, *confidence*, whose value is an integer from 1 to 5 representing the model's confidence in its answer (in the value of *missingSteps*). Here 5 indicates perfect confidence, 1 is zero confidence, 2 is some (low) confidence, 3 is medium confidence, and 4 is strong but not full confidence.<sup>93</sup>

Table 30 shows some elementary statistics about the gap-filling data along 5 dimensions: the original length of each of the 200 proofs (before the insertion of gaps); the size of each gap (the number of steps removed from the original proof); the size of each equational theory (the number of equations in each of the 200 problems); the number of equations cited per proof step; and the sizes of the equations themselves, viewed as ASTs. As the table shows, the average gap is slightly longer than half the size of the proof, so these are quite substantial reasoning holes.

The main metric we report for this task is accuracy, the ratio  $\#(\text{correct answers}) / \#(\text{all answers})$ . When is a gap-filling answer considered correct? If the value of *missingSteps* is the empty list of steps, then this is correct iff the gap cannot be filled.<sup>94</sup> If the value of *missingSteps* is a non-empty list of steps specifying a subproof, it is considered correct iff the subproof closes the gap correctly in the sense specified by (15). We use our own code to decide, for any given  $s$ ,  $t$ , and subset  $S \subseteq \mathcal{E}$ , whether  $s \iff_S t$ .

<sup>93</sup> In earlier versions we had requested confidence levels to be continuous numbers from 0 to 1, but in practice models would only output numbers in a small fixed set, such as  $\{0.2, 0.4, 0.6, 0.5, 0.8, 0.9, 0.95\}$ , so we moved to an inherently discrete formulation.

<sup>94</sup> This is determined with the help of an ATP (automated theorem prover).

Quantity	Mean	SD	Median	Min	Max	Spark
Proof length	18.91	9.59	18.50	3	35	
Gap length	11.72	7.07	11.5	1	31	
Equations per problem	24.84	12.51	24.00	3	53	
Equations cited per step	1.29	0.61	1.0	1	3	
Individual equation size	8.59	4.02	8.00	4	40	

**Table 30.** Data distribution summaries with inline box–sparklines for the gap-filling task slice of EQ<sub>1</sub>. Box = IQR (Interquartile Range); red line = median; dot = mean; whiskers = 5–95th.

Before we present the results, it is interesting to see how many proofs were written by each model (as opposed to returning an empty list of missing steps, indicating that the gap is unfillable), and how long these proofs were. The table below shows the number of proofs written:

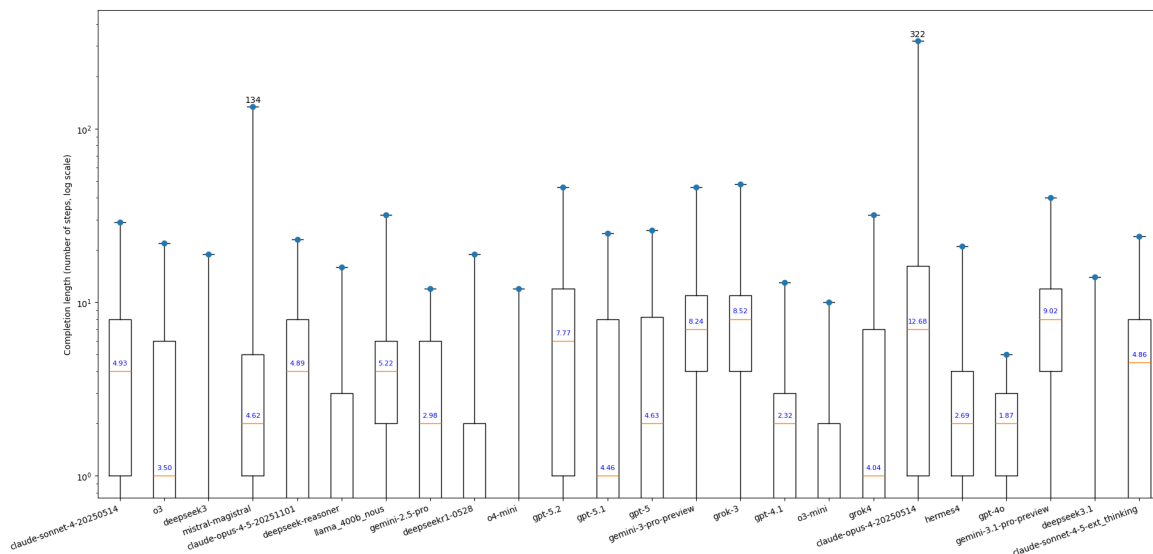
The length distribution for these proofs is shown in Fig. 42. Recall that the gaps themselves have median and mean sizes between 11 and 12, with 95% of all gaps having no more than 23 steps. GPT-5 and Grok-3/Grok-4 are the models that come closest to that distribution. Most other models fill the gaps with completions that are either too short (GPT-4o) or too long (Claude Opus 4.1, which produced a subproof with 322 steps that is not only incorrect but also goes in circles, repeatedly reiterating many of the same steps). Claude Opus 4.1 is not the only model to write proofs with cycles: 64 of the 154 proofs written by Mistral Magistral have cycles, 16/138 for Claude Opus 4.1, 19/194 for Grok-3, 4/140 for GPT-4.1, 4/146 for Claude Sonnet 4, 3/44 for DeepSeek-3, 42/199 for Llama-3.1-405b, 1/114 for GPT-5 and 1/104 for Grok-4. The rest of the models did not have any cycles in their proofs. Note, however, that these absolute counts are imperfect measures of duplication given the length distribution in Fig. 42, since some models consistently write much shorter proofs and thus have fewer opportunities to introduce cycles.<sup>95</sup>

The main results can be found in Table 32. Note the large performance gap between GPT-5.1 and GPT-5.2, and the even larger difference between Gemini 2.5 Pro and Gemini 3 Pro Preview. The DI (discriminability index) of these results is 0.26.

<sup>95</sup> A more appropriate redundancy measure would be the number of non-unique steps divided by the total number of steps in the proof.

Model	Proofs Written	Model	Proofs Written
deepseek-v3.1	43	deepseek-v3	44
o4-mini	49	deepseek-r1-0528	63
deepseek-reasoner	67	o3-mini	86
grok-4	104	o3	104
gpt-5.1	105	gemini-2.5-pro	111
gpt-5	114	claude-opus-4-5-20251101	129
claude-sonnet-4.5-ext-thinking	132	claude-opus-4-20250514	138
gpt-4.1	140	claude-sonnet-4-20250514	146
gpt-5.2	153	hermes-4	153
mistral-magistral	154	gpt-4o	183
gemini-3.1-pro-preview	183	gemini-3.0-pro-preview	186
grok-3	194	llama-3.1-405b	199

**Table 31.** Number of nonempty proof completions written for the gap-filling task.



**Figure 42.** Completion length distribution by model (using boxplots and log scale).

Fig. 43 shows a classification of all possible errors, assuming structurally correct responses by the models. There are five main error types:

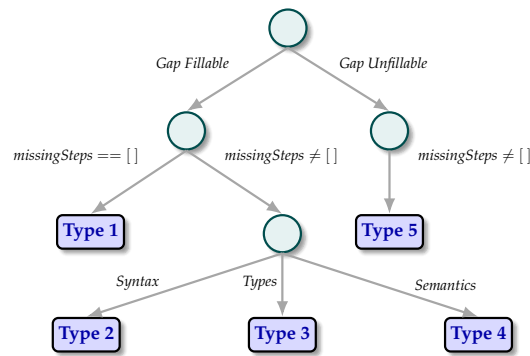
1. *Type 1 errors*: the gap is fillable but the model outputs an empty completion, signaling that the gap cannot be filled.
2. *Type 2 errors*: the gap is fillable and the model outputs a non-empty completion, but (at least) one of the terms in the proof has unbalanced parentheses (a “syntax error”).
3. *Type 3 errors*: the gap is fillable and the model outputs a non-empty completion, but (at least) one of the terms in the proof has a “type error,” meaning that a function symbol is applied to the wrong number of arguments.
4. *Type 4 errors*: the gap is fillable and the model outputs a non-empty completion, but the proof does not go through because a step is incorrect, in that the cited equations do not justify the derivation of the relevant term.
5. *Type 5 errors*: the gap is not fillable, but the model outputs a non-empty completion that claims to fill it.

These can be partitioned into 3 disjoint blocks,  $\{1\}$ ,  $\{2, 3, 4\}$ , and  $\{5\}$ . Types 2, 3, 4, are not mutually exclusive because a proof can have syntax errors as well as type errors and/or semantic errors.

Almost all responses by all models were structurally sound (with a sole exception from llama-3.1-405b, one of whose proofs had a step without the required *supportingEquations* key). Fig. 44 shows raw counts for the five error types for all models.

Model	Accuracy	Model	Accuracy
gemini-3.1-pro-preview	0.82 ( $\pm 0.05$ )	gemini-3.0-pro-preview	0.62 ( $\pm 0.07$ )
gpt-5.2	0.61 ( $\pm 0.07$ )	gpt-5	0.49 ( $\pm 0.07$ )
gpt-5.1	0.49 ( $\pm 0.07$ )	grok-4	0.48 ( $\pm 0.07$ )
claude-opus-4-5-20251101	0.41 ( $\pm 0.07$ )	gemini-2.5-pro	0.40 ( $\pm 0.07$ )
o3	0.32 ( $\pm 0.06$ )	deepseek-reasoner	0.27 ( $\pm 0.06$ )
claude-sonnet-4.5-ext-thinking	0.18 ( $\pm 0.05$ )	claude-opus-4-20250514	0.17 ( $\pm 0.05$ )
deepseek-r1-0528	0.15 ( $\pm 0.05$ )	o4-mini	0.15 ( $\pm 0.05$ )
claude-sonnet-4-20250514	0.07 ( $\pm 0.04$ )	deepseek-v3.1	0.06 ( $\pm 0.03$ )
deepseek-v3	0.06 ( $\pm 0.03$ )	o3-mini	0.05 ( $\pm 0.03$ )
mistral-magistral	0.04 ( $\pm 0.03$ )	hermes-4	0.03 ( $\pm 0.03$ )
llama-3.1-405b	0.03 ( $\pm 0.03$ )	gpt-4o	0.03 ( $\pm 0.03$ )
grok-3	0.01 ( $\pm 0.02$ )	gpt-4.1	0.01 ( $\pm 0.02$ )

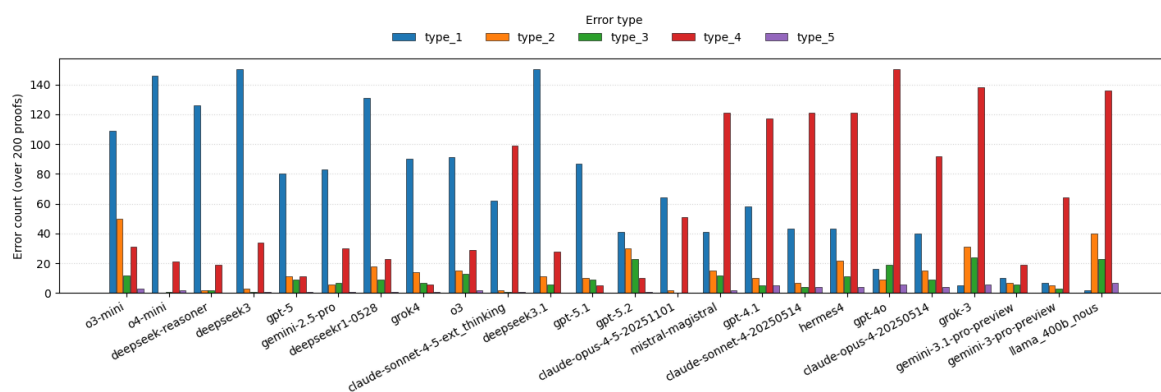
**Table 32.** Gap-Filling accuracies.



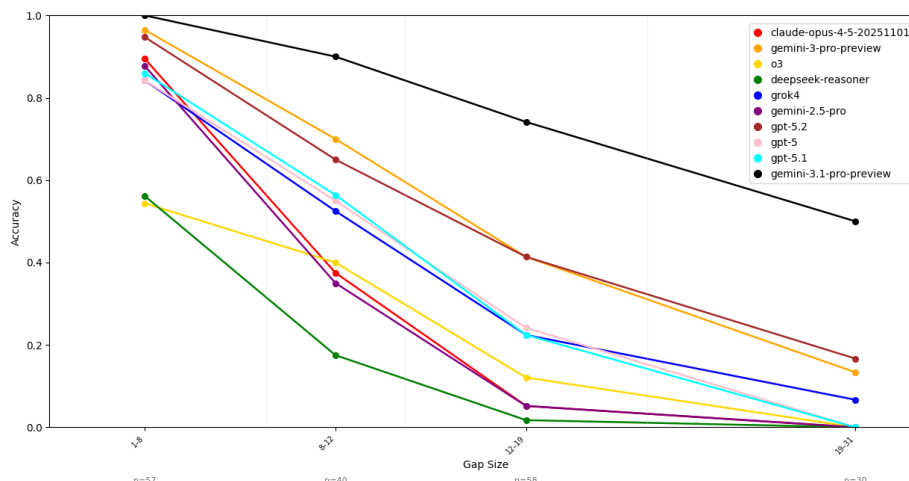
**Figure 43.** Error typology for the gap-filling task.

Some high-level takeaways from these findings:

- The large majority of gaps (193 out of 200) are fillable, so type-5 errors are rare.
- As seen in other tasks, when failure-averse models cannot solve a problem, they prefer to fail succinctly rather than generate longer answers, which means they tend to produce the smallest possible output that adheres to user specifications. DeepSeek models lead the pack here. DeepSeek-V3 and DeepSeek-3.1, for example, refuse to fill 156 and 157 out of the 200 gaps, respectively, claiming that they cannot be filled. However, this should not be interpreted as the models being aware of their limitations. If the models had such awareness, they would generate these answers with low confidence. By contrast, both DeepSeek-V3 and DeepSeek-3.1 (for example) wrongly claim that more than 150 fillable gaps are unfillable with a very strong median confidence of 4. (DeepSeek-V3 does not even once output a “very low” confidence of 1.) In their explanations, these models often admit that they arrived at a negative verdict due to the problem’s complexity (a favorite phrase is “Given the time constraints and complexity, I concluded that the gap cannot be filled validly”) and yet they claim a high confidence in the verdict nevertheless, as if problem complexity by itself was evidence of unsolvability. Likewise for R1 and OpenAI’s mini models, although o4-mini is much more humble in its confidence.
- By contrast, the “compulsively generative” models with the tall red bars on the right side of the figure have few type-1 errors because they tend to plunge ahead and fill the gaps even if they’re not in a position to do so successfully. As a result, the vast majority of errors for these models are of type 4, genuinely failing proof steps.
- Stronger models also tend to provide better-calibrated confidence levels. Here is an explanation provided by Gemini-2.5-Pro: *The transformations required, particularly for the first and fourth arguments of the main term, are exceptionally complex. Despite significant effort, I could not construct a complete and verifiable path from start\_term to t\_k. The search space of possible rewrite sequences is vast due to the large number of equations. Given the difficulty in finding a valid path and the high chance of error in such a complex*



**Figure 44.** Prevalence of errors of different types in the gap-filling task.



**Figure 45.** Gap-filling accuracy for the top 10 models as a function of gap size.

derivation, I am unable to provide a sequence of steps with high confidence. It is possible that a key insight or a simpler path was missed, but it is also possible the gap is invalid as stated. Therefore, I will indicate that the gap is invalid by providing an empty list, but with low confidence, reflecting the possibility that a solution exists but is beyond my current analysis capabilities. That said, our ECE analysis below shows that all models are consistently over-confident, and Gemini-2.5-Pro is the worst-calibrated of the elite models.

- Syntax and type errors (types 2 and 3, respectively) are less common, but they do occur in non-negligible numbers across all models, including the very best. Their presence indicates that the models' internal representation of these symbolic structures is not quite robust. They can still lose a parenthesis or misapply a function. The high rate of syntax errors by o3-mini is particularly noteworthy—it makes such errors in a remarkable 50 proof attempts, or 25% of all inputs.
- Regarding instruction following, recall that the prompt explicitly requires all steps to cite no more than two equations. While virtually all models adhere to that constraint at the 95<sup>th</sup> percentile (the only exception is DeepSeek-V3, which cites 2.3 equations per step even at the 90<sup>th</sup> percentile), only a minority of the models (10 out of 24) achieved perfect adherence to the constraint, without ever citing more than 2 equations: GPT-5.2, GPT-5.1, GPT-5, Gemini-3.1, Claude Opus 4.5, Grok-3, o4-mini, o3-mini, DeepSeek-3.1, and DeepSeek-Reasoner. For all other models, the maximum number of cited equations on a given step ranges from 3 (for Grok-4) to 10 (for GPT-4.1).
- The best models clearly show an ability to compose nontrivial reasoning chains. For a problem with 46 equational axioms and a gap size of 23, Gemini 3 Pro Preview managed to fill the gap with a subproof of 35 steps, each one citing a single equation. However, this was the longest correct proof written by Gemini 3 and an outlier. As we will see shortly, performance drops sharply across the board as the gap size increases.

We now examine how accuracy depends on gap size as well as the size of the equational theory for the top 10 models. These two relationships are depicted in Fig. 45 and Fig. 46, respectively. The plots show that as the problems become more challenging, performance drops drastically even for the strongest models.

In a sense, gap size captures the depth of the required search while theory size captures its breadth. The size of the gap is a measure of the number of needed reasoning steps, which taxes the model's ability to maintain a goal, avoid getting lost, and chain inferences correctly. By contrast, the number of equational axioms taxes the model's ability to select the *correct* tool at each step from a large set of distractors. It's a measure of its ability to handle a large context of rules.

The heat map in Fig. 47 shows how model accuracy (across all models) depends jointly on both parameters: gap size on the  $x$ -axis and equational theory size on the  $y$ -axis. Each cell's color represents the fraction of correct proofs among all attempts falling into that combination of gap size and theory

size, with lighter colors indicating higher accuracy. The pattern reveals a clear gradient: accuracy is highest in the bottom-left corner, where gaps are small and theories are simple, and drops sharply as either dimension increases. Moving to the right, accuracy decreases with larger gaps; moving upward, it decreases with larger theory sizes, even as gap sizes stay small. In the top-right region, where both the gaps and the theories are large, accuracy is near zero, showing that these two sources of difficulty combine to severely degrade model performance. Overall, the heat map highlights that while both factors independently hurt success rates, their effects compound. Models that can handle a relatively large gap with a small theory, or conversely, break down almost completely when both are large. This shows that current models lack the robust search and planning abilities needed to navigate complex combinatorial search spaces effectively.

The plot in Fig. 48 compares the calibration of the models on the same dataset of 200 proofs. For each model, 6 thin bars summarize calibration performance. The first five bars (L1–L5) correspond to absolute calibration gaps in each of the model’s discrete confidence levels. Recall that we allow for 5

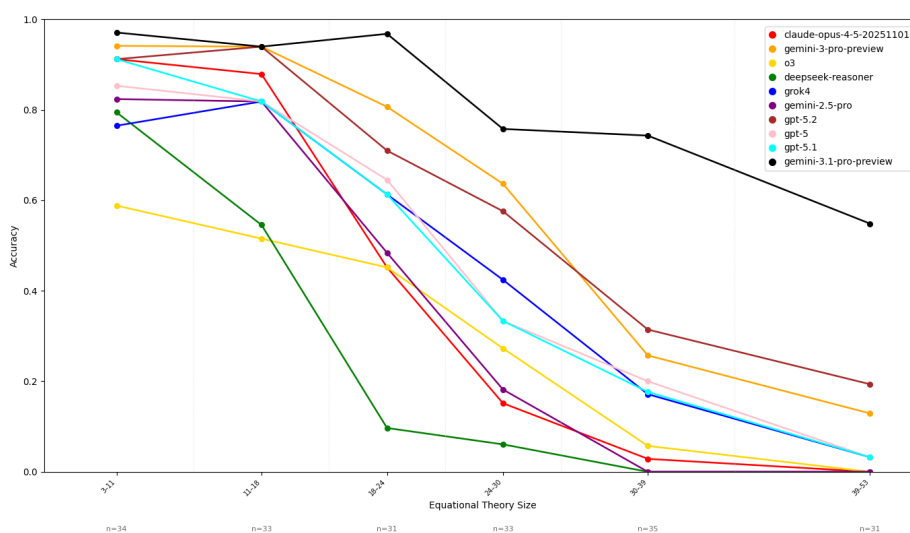


Figure 46. Gap-filling accuracy for the top 10 models as a function of equational theory size.

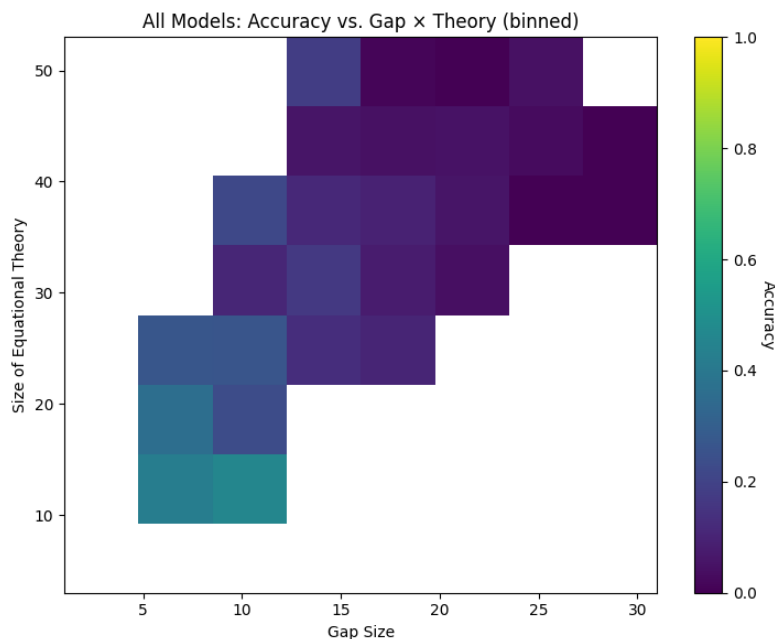
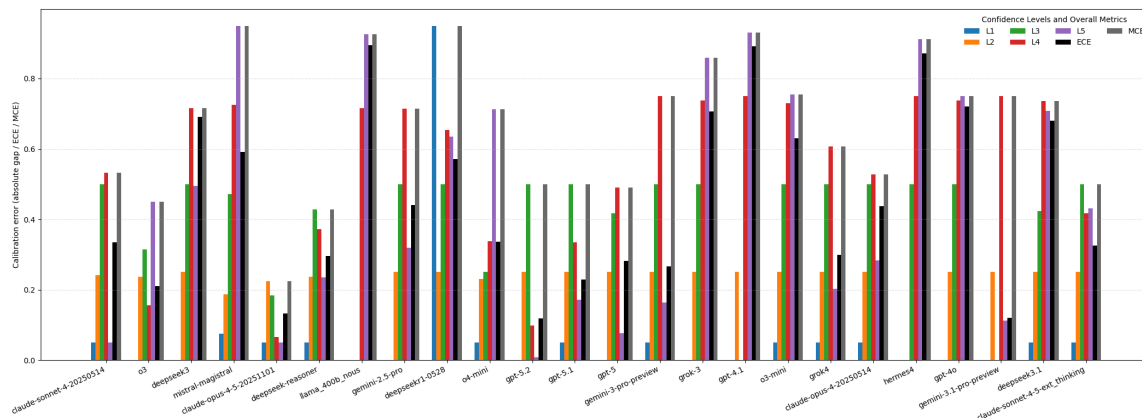


Figure 47. Consolidated accuracy heat map for gap filling, as a function of gap length and equational theory size.



**Figure 48.** Gap-filling calibration errors across all models (bin-level and overall ECE).

possible confidence levels, 1 (very low confidence) through 5 (near certainty). We map these levels to probabilities as follows:

$$\{1 \mapsto 0.05, 2 \mapsto 0.25, 3 \mapsto 0.5, 4 \mapsto 0.75, 5 \mapsto 0.95\}. \quad (16)$$

The gap for a given bin is the absolute value of the difference between the empirical accuracy observed in that bin and that predicted by the probability assigned to that bin. For instance, the probability assigned to bin 2 (for a confidence level of 2) is 0.25, so if a model were perfectly calibrated we would expect its outputs to be correct 25% of the time. If they are only accurate 10% of the time, we have an absolute gap of 15%—a significant marker of overconfidence. In the plot, the height of each bar shows how far the model’s confidence in that bin differs from its actual empirical accuracy: a higher bar means worse calibration. The final, black bar shows the overall Expected Calibration Error (ECE), a weighted average across bins, with the following standard definition:

$$ECE = \sum_{m=1}^M \frac{|B_m|}{n} \cdot |acc(B_m) - conf(B_m)|,$$

where  $M$  is the number of bins (5 in our case);  $B_m$  is the set of predictions that fall into bin  $m$ ;  $n$  is the total number of predictions (200 in our case);  $acc(B_m)$  is the fraction of correct predictions in  $B_m$ ; and  $conf(B_m)$  is the average confidence score of the predictions in the bin.<sup>96</sup> MCE is defined as the maximum calibration error across all bins:

$$MCE = \max_{m \in \{1, \dots, M\}} |acc(B_m) - conf(B_m)|.$$

These are the two canonical calibration metrics that are most commonly reported.

Looking across all models, none of them are well-calibrated. Only GPT-5.2, Gemini-3.1, and Claude Opus 4.5 (with average errors of 0.12, 0.12, and 0.13) land in a range that most calibration researchers would consider respectable. While not ideal, it is small enough that the models’ confidence estimates could plausibly carry some actionable signal. All other ECE scores are above 0.2, which is generally considered high miscalibration. Everything above 0.4 (this includes Gemini 2.5 Pro, o3-mini, and DeepSeek-R1-0528) is basically noise, while some models (e.g., mistral-magistral, llama-3.1-405b, and grok-3) are essentially completely uncalibrated, with extremely large per-bin errors ranging from 0.7 to 0.9. The stated confidence of these models is usually off by 70-90 percentage points, which is not mere miscalibration but complete epistemic collapse. The worst bins across almost all models are L4 and L5, which correspond to predictions made with high or very high confidence. This is classic

<sup>96</sup> This is the general definition of ECE. In our setting, because we don’t have continuously varying probabilities,  $conf(B_m)$  is fixed for each bin by the mapping (16).

overconfidence: models severely overestimate their reliability when they are confident. In other words, the models are most wrong precisely when they think they're most right, which is the opposite of what is needed in mathematical reasoning. If confidence levels cannot be trusted specifically when they are high, they are of little value as guardrails, e.g., thresholding on "confidence  $\geq 4$ " would be unsafe for all of these models.<sup>97</sup> The exception is GPT-5.2, which has an exceptionally small error in L5, 0.008. Claude 4.5 Opus also has a small calibration error in L5, 0.05. This suggests that when these two models assign near-maximum confidence, their predictions are very likely correct. Gemini-3.1 has a more sizable L5 error, 0.12. Note, however, that both GPT-5.2 and Gemini 3.1 still exhibit severe miscalibration in other bins, with very high MCE values (0.5 and 0.75, respectively), leaving significant room for improvement. At 0.22, the MCE of Opus 4.5 is lower but still large.

These results are interesting for a couple of reasons. First, they show that miscalibration is not inevitable; it can be dramatically reduced. Second, combined with the accuracy results, they show that calibration is not a corollary of correctness. While accuracy and calibration do tend to increase together, the correlation is far from perfect. For example, Gemini 3 is the second best-performing model but it is miscalibrated, featuring a significant ECE of 0.27—more than twice that of GPT-5.2. Conversely, Claude Opus 4.5 is much better calibrated but its accuracy is only 41%. Therefore, calibration is not an emergent property of scale and reasoning competence alone. It very likely requires explicit uncertainty supervision and/or architectural or training changes that link reasoning depth to confidence. The three best calibrated models almost certainly had an additional training signal or mechanism binding confidence to correctness. That could have been explicit calibration loss, confidence supervision, rejection-aware training, or tighter coupling between reasoning depth and confidence output.

## 10. Conclusions

This paper introduced PROOFGRID, a new benchmark designed to assess the reasoning abilities of LLMs through tasks that demand rigorous and machine-checkable proof construction and analysis under minimal formalism. Unlike benchmarks that focus solely on answer accuracy, PROOFGRID emphasizes the integrity of a model's inferences and offers a more precise and fine-grained assessment of reasoning competence.

A core theme emerging from our results is one of extraordinary progress at a rapid pace. We started this work in late 2024. At that point, the state of the art was GPT-4 and Claude Sonnet 3.5, which were essentially incapable of sustained reasoning. We were concerned at the time that what are now our easiest tasks, **PL1-PW** and **PL1-PC**, might be too challenging; performance was well below ceiling and overrun with elementary errors. Within a year, these tasks had become easy for the best models and accessible even to mid-tier models. Our current model results provide many examples of superb fluid reasoning on highly challenging abstract problems, suggesting that the best models have developed powerful general inferential strategies. The debate as to whether these systems can reason at all should be considered settled at this point: they can. The evidence in our results is overwhelming. PROOFGRID should provide a robust platform for tracking and accelerating these developments, underscoring the importance of proof-based evaluation as models move toward deeper integration into scientific and mathematical workflows.

Nevertheless, major challenges remain. Even the best models continue to make rudimentary reasoning errors (albeit at a much lower rate than before), and all models quickly run up against hard cognitive limitations as the difficulty of the reasoning increases, especially on structurally rich problems drawn from classical families of hard problems. More concerningly, we see high degrees of what we call *epistemic instability*: the tendency of models to "forget" what they seem to know very well when they are working on a hard problem. We formalized this phenomenon by introducing a new

<sup>97</sup> To put it in terms of odds, an ECE error of 0.25 in bin 5 means that when the model says *I'm  $\approx$  95% sure I'm right*, it's actually right only about 70% of the time. Taking the model at its word implies betting odds of 19:1, i.e., you expect to win (the model to be right) about 19 times out of 20. But the real odds are 7:3, so you'd win only about 7 times out of 10. If you are willing to risk \$19 to win \$1 because you believe the 19:1 odds (you take the model at its word), then the expected value of your betting strategy per decision would be  $0.7 \cdot 1 + 0.3 \cdot (-19) = -5$ , i.e., on average you would lose \$5 per decision.

quantity called ESI (epistemic stability index) and we presented the first set of empirical results on that front.

In particular, while our results strongly suggest that models possess at least partial reasoning competence, they still fall short on procedural grounding and cognitive control, which often prevents them from *rigorously* executing reasoning tasks without external scaffolding. By and large, they reason in a semantic space, not a formal one. They often identify which conclusions follow from which assumptions and construct globally coherent proof strategies. However, they fail to reliably execute these strategies within a formally constrained system, even a very simple one. They struggle to suppress invalid but tempting operations, to adhere strictly to syntactic constraints, and to treat inference rules as deterministic operators rather than as semantic heuristics. In that sense, current models exhibit semantic reasoning without procedural grounding.

The issue can perhaps be illuminated through the lens of the manifold hypothesis (Bengio et al. 2013). Neural models perform well when the data distribution lies on a low-dimensional manifold embedded in a high-dimensional representation space, as is the case for images or natural language. In such domains, nearby points in representation space typically correspond to semantically similar objects, and interpolation is meaningful. Formal logical reasoning appears to violate this assumption. The space of formal proofs is extremely sparse, and small perturbations to a formula or an inference rule often result in radical and disruptive semantic changes rather than smooth variation. Unlike natural language, where many paraphrases express essentially the same meaning, formal proofs do not allow for much redundancy: a single misplaced connective, rule application, or even a stray parenthesis can invalidate the entire derivation.

From this perspective, the behavior we observe is unsurprising. Models trained to operate in smooth semantic spaces attempt to solve proof tasks by interpolating within a semantic manifold that, in this domain, does not meaningfully exist. This results in “semantic” reasoning that often captures the intended structure of an argument but fails to satisfy the discrete, brittle constraints imposed by formal proof systems. In this sense, current models exhibit reasoning that is well-adapted to continuous semantic domains but poorly matched to the sharply discontinuous geometry of rigorous, formal proofs. Their failures may therefore reflect a deeper mismatch between the inductive biases of deep learning and the structure of symbolic reasoning. This is not to say that formal reasoning is inherently non-manifold or that DL cannot in principle approximate it well enough; only that, at least in their current training and architectural incarnations, the continued symbolic brittleness of models may be structurally related to the apparent disconnect between their inductive bias and the geometry of formal reasoning spaces.

Of course, the rapid improvement of frontier systems suggests that this geometric tension is not insurmountable. But it is worth noting that contemporary high-performing models are no longer simple next-token predictors operating in isolation; many incorporate inference-time search, reranking, verifier guidance, or other forms of structured deliberation, and often distill the results of such search back into their parameters. These mechanisms effectively introduce discrete branching, pruning, and constraint enforcement steps that differ qualitatively from token-by-token generation and partially compensate for the smooth, interpolation-driven inductive biases of the underlying neural representations. In this sense, the gains we observe may not so much reflect the disappearance of symbolic brittleness, but the increasing use of architectural guardrails that approximate modular, search-based reasoning within an otherwise continuous semantic substrate.

A related recurrent pattern has been failures of cognitive isolation or computational modularity. To put it simplistically, with transformer-based architectures it is not the case that “if a model can reliably solve task  $S$  and task  $T$  is composed of many  $S$  subtasks, the model can also reliably solve  $T$ .” Reliable problem solving needs reliable context-switching into a cognitively isolated “system-2 mode” that can load a particular sub-problem’s details into working memory while shutting out everything else as irrelevant noise. This is what both humans and conventional symbolic computers do: they use function calling to encapsulate complexity. When a function is called, its internal logic is independent

of and isolated from the global state of the program, which can be arbitrarily complicated without affecting the execution of the function. In a transformer model, by contrast, every token attends to every other token in the entire context window. In checking PL<sub>3</sub> proofs, for example, the embeddings for the tokens in step 42 are mathematically influenced by the attention patterns from all previous steps and the system prompt. So the model's mini-SAT circuits that are used to solve the local entailment problems in such a proof are not running in isolation; they are running in a big noisy room that subjects the output to interference from irrelevant sources. Our results underscore that LLMs do not draw upon modular internal reasoning capabilities; if they did, their performance would be much more epistemically stable and invariant to context. Instead, they perform a context-dependent simulation of reasoning, essentially approximating system-2 behavior using mechanisms designed for system-1 tasks. That simulation degrades when the model's architectural and representational resources (such as effective attention bandwidth and feature superposition limits, respectively) are stretched thin by maintaining the global reasoning state.

These findings suggest that proof-based evaluation is not merely a stricter grading scheme, but a lens into the structure of current reasoning systems. The gap between semantic fluency and formal control reveals where architectural biases still dominate. In that sense, PROOFGRID is less a leaderboard and more a diagnostic instrument for the evolving nature of machine reasoning.

We close with a brief **FAQ** that we hope will help to clarify and reinforce some of this paper's key points.

- **Q1:** Why use formal notation? Forcing models to express their reasoning in a formal language like NDL introduces confounds, such as artificial syntax and formatting errors.

**A:** From first principles, there are only two main options for a reasoning benchmark: models can express their reasoning either in a formal language or in natural language.

For testing purposes, the core foundational problem with natural language is that it doesn't scale. There is no algorithm for determining the correctness of an arbitrary stretch of text that purports to reason its way to a conclusion. As a result, natural-language reasoning cannot be reliably and *automatically* evaluated, and must instead be manually inspected. This difficulty explains why most existing "reasoning benchmarks" do not actually evaluate the reasoning itself (ignoring, for example, COT traces during scoring), because there is no procedure that can take an arbitrary natural-language reasoning trace and reliably output *correct* or *incorrect*. This limitation alone is reason enough for AI benchmarks to move away from natural language and toward formal languages, which make reasoning amenable to computational analysis and evaluation.

Moreover, natural language doesn't remove confounds; it simply obscures them. In plain English, we can only determine whether a reasoning trace looks plausible. Assumption scope is a simple case in point. In a formal language, this is explicitly indicated with opening-closing braces (or **begin-end** pairs, Python-style indentation, etc.), leaving zero doubt about what hypotheses are active and when. In English, this has to be guessed from the text by taking into account a wide range of signals, including formatting details like the way in which the text is broken into paragraphs. Experienced authors write in a way that minimizes guesswork, but ambiguity often remains: "Suppose that  $p$  holds. Then . . . . Next, consider  $q$ . We have . . ." Is  $p$  still active inside the second ellipses? What if there is a new paragraph start right before "Next, consider  $q$ ."?

It might be argued that natural-language reasoning *can* be mechanically evaluated, using "LLMs as judges" Zheng et al. (2023). But there is an air of circularity in that approach. Who has judged the judges, and where can we look for independent ground truth? Using a system of unknown or dubious reasoning reliability to judge reasoning correctness seems questionable. An NDL proof can be independently evaluated by a proof checker that has a perfectly precise specification, is written in a few pages of code that can be audited or even formally verified, and does not change across runs, resulting in an exceptionally small and stable TCB (trusted computing base). An LLM judge, by contrast, has billions of parameters, cannot be easily audited, has no formal semantics or

specification, and its behavior may depend on temperature, prompt, and version changes, among other factors.

Finally, the use of a formal language like NDL provides much higher analytical resolution, allowing us to *measure* cognitive control. We are able to precisely distinguish parsing errors from malformed rule applications, wrong conclusions, purely logical errors, unbound identifiers, and so on. Such fine-grained analysis is simply not possible in natural language, where most issues collapse into “the model’s answer seems wrong.” Formal languages, in other words, make deviations from correct reasoning explicit and mechanically detectable, whereas natural-language reasoning often obscures such failures behind linguistic ambiguity. Models can “sound right” while being wrong, and logical gaps can easily be hidden in rhetorically compelling prose. In this sense, the errors we observe when models use NDL are not confounds but diagnostic signals, revealing limitations in models’ ability to reliably execute rigorous reasoning. A natural-language formulation would avoid some of these surface errors, but at the cost of making correctness judgments substantially less precise and more subjective, and error analysis much more brittle and qualitative.

- **Q2:** There is a third, middle-ground alternative: controlled natural language (CNL). Why not require models to express their reasoning in a controlled fragment of English that can then be formally translated into logic and computationally analyzed?

A: CNL sounds like the best of both worlds (natural and formal languages), but in this setting it would cause greater difficulties. Crucially, CNL here would not be given as *input* to the models (as is widely done, for example, in benchmarks like RuleTaker (Clark et al. 2021), see Section 2), but would instead be required as the *output* of the models. This immediately presents two problems. First, the CNL must be formally defined and its entire spec must be given to the model, so that the model knows exactly what it is allowed to say and what sort of sentences are off limits. The definition would need to fully specify allowable sentence structures, anaphoras, and tense and voice; the interpretation of conjunctions; the expression of implications; scope rules; the introduction and discharge of assumptions; what counts as a valid reference; how to disambiguate ambiguous constructions; and so on. This would be orders of magnitude harder than specifying NDL. The issue is that, for all practical purposes, any CNL is itself a formal language. It must commit to a grammar, a vocabulary, scoping and disambiguation rules, an interpretation function, and so on. The main difference is that its syntax is less precise, its semantics are harder to specify, and its failure modes are less transparent. By contrast, a language like NDL has a tiny grammar (it can be written in BNF with a handful of productions) and its operational semantics can be fully described in a page or two of text.

Second, CNL would not eliminate issues of procedural control. There are no guarantees that models would respect the formal spec and reliably remain within the bounds of the prescribed fragment, just as they may fail to respect the rules of NDL. Models would likely be “almost compliant” while drifting outside the allowed fragment, as a single stray clause, pronoun, or tense change can break parsability. With NDL, we can detect and repair or overlook minor deviations, as we have done with our instrumented proof checker. Achieving the same level of robustness for a CNL would be much more challenging.

- **Q3:** Why NDL? NDL has hardly ever been seen by models during their pretraining. Isn’t it unfair to make them reason in what is essentially a foreign language to them, and aren’t a lot of the model errors you’ve encountered due to this unfamiliarity? Why not use a language like Lean or Isabelle, which are much more extensively represented in a typical pretraining corpus?

A: A reasoning benchmark should be universally applicable to any model, without requiring prior exposure to a specific formal system. In fact, if we are serious about testing fluid reasoning, we should minimize reliance on the sort of crystallized knowledge and skills that accrue when a model ingests large amounts of pretraining material that it can directly leverage at test time. We want to measure reasoning ability, not tool familiarity. If a model succeeds in Lean because it

saw a lot of it in training and has therefore memorized common patterns and tactic heuristics, then we are testing recall and imitation just as much as we are testing reasoning. We should be testing whether a model can internalize and follow a system of rigorous rules from specification, not whether it has memorized one. For testing purposes, the absence of strong NDL priors is an advantage, not a liability.

Another point (already made in Section 2, but it bears repeating here) is that Lean proofs are procedural and tightly coupled with proof search in a sequent calculus. This is unsurprising, given that Lean is an interactive theorem-proving system and thus naturally more focused on tactic orchestration and automation steering. NDL, by contrast, is more focused on describing the proofs themselves, rather than the search for them. It mirrors natural deduction, expresses proofs entirely declaratively, makes assumption and logical scopes explicit, and separates logical structure from proof search.

Further, as we also already pointed out in Section 2, languages like Lean are intentionally powerful and thus correspondingly complex, featuring esoteric syntax forms, heavy type theory, opaque automation, and a large ecosystem of libraries, modules, and built-in theories. This makes Lean very expressive, but also unsuitable as a fruit fly for rigorous reasoning. NDL, by contrast, is intentionally minimal. It fits fully in a short prompt, has no hidden machinery or inference automation, no meta-level tactics, and no library dependencies. This makes it cleaner, simpler, entirely self-contained, tooling-independent, much more diagnostic (because its simplicity exposes real inferential limitations), and much more appropriate as a controlled testbed for reasoning.

Finally, with respect to the fairness concern, we restate our earlier position: the main role of a benchmark is to enable meaningful comparison across models, not to extract their maximum achievable performance. A benchmark is a diagnostic instrument, not an exercise in system optimization. Fairness is ensured by clearly specifying the evaluation criteria and applying them consistently to all models.

- **Q4:** Why instrument the proof checker? Doesn't that help the models to cheat?

**A:** Instrumentation has been a sharp analytical tool. While NDL may be exceptionally simple, it is still a formal language. As a result, strict checking of NDL proofs would flatline the accuracy of many models at exceedingly low levels, often due to minor procedural slips. This would collapse multiple failure modes into a single low score, yielding no discrimination and no information about latent reasoning competence. By contrast, our instrumented proof checker reveals a graded competence spectrum, large separation between models, and internally coherent error profiles that correlate with other behavioral indicators. Far from helping models cheat, instrumentation restores measurement resolution and disentangles executive brittleness from higher-level reasoning.

- **Q5:** You claim that using abstract symbolic problems allows you to test fluid reasoning because the problems don't have any content (other than pure logical content), and so the models cannot leverage their huge knowledge banks to retrieve memorized answers. However, the pretraining of modern models exposes them to a vast range of proof templates, patterns of symbolic manipulation, typical lemma structures, and so forth. This means that even abstract tasks can partially reduce to pattern reuse. Doesn't it follow then that PROOFGRID-style benchmarks test the ability to generalize reasoning strategies absorbed during pre- (and post-) training, rather than tabula-rasa reasoning?

**A:** That's a valid point, and we agree with it in part. Our claim is not that PROOFGRID measures tabula-rasa or psychometrically pure fluid reasoning in any strong sense. Contemporary models plainly rely on pretraining to acquire the concepts of proof, logical entailment, symbolic manipulation, and deductive structure. In that respect, PROOFGRID tasks (like essentially every benchmark for advanced cognition) tests the ability to deploy and generalize previously learned reasoning strategies, rather than reasoning ex nihilo. The point of PROOFGRID is narrower and more concrete: it probes a model's native analytical competence in symbolic reasoning, partic-

ularly deductive reasoning, and does so in a small, simple, and fully specified formalism, on abstract problems whose solutions are extremely unlikely to be directly memorized. Because the problems are content-free and procedurally constrained, the model cannot rely on world knowledge, standard textbook proofs, or domain-specific theorem memories. It must instead instantiate general proof skills in a new setting. So the benchmark is best understood not as a measure of pure fluid intelligence, but as a measure of a model's ability to compose, verify, analyze, and manipulate proofs under minimal semantic scaffolding.

It is also worth reiterating that all nontrivial cognitive benchmarks test the reuse of learned strategies. Human IQ tests do too. Nobody reasons from nowhere. Humans bring years of schooling, language, symbol use, and learned problem schemas to Raven matrices, analogies, and logic puzzles. The relevant question is not whether prior learning matters, but what kind of prior learning is useful and how directly task success can be reduced to retrieval. It may be useful to distinguish three different phenomena: (a) memorization of specific proofs; (b) reuse of learned proof methods; and (c) the ability to coordinate these methods correctly during reasoning when tackling a novel problem. The symbolic nature of PROOFGRID helps to rule out (a): simple regurgitation of famous contentful proofs. This is what we would get if we were just asking for proofs of the infinity of primes or the irrationality of the square root of 2. PROOFGRID is not robust against (b), nor should it be. If a model has learned useful proof techniques, such as proof contradiction and case analysis, that is part of what mathematical competence consists in. And what PROOFGRID probes quite effectively is (c): whether a model can compose these techniques correctly in a rigorous setting.

## 11. Limitations

While PROOFGRID offers a wide-ranging and deep evaluation of logical reasoning capabilities in language models, a number of limitations should be acknowledged.

First, PROOFGRID intentionally tests logical reasoning in (minimally) formal languages, not natural-language reasoning. Thus, PROOFGRID's results may not fully generalize to models' reasoning abilities in natural language contexts. However, as we argue in the paper, tests of natural-language reasoning are outcome-based and thus only gauge reasoning competence indirectly.

Second, our evaluation metrics focus on accuracy rather than efficiency. Different models may achieve similar accuracy scores while varying significantly in test-time compute or other resource requirements. A more comprehensive evaluation would include these dimensions.

Third, while propositional and equational logic are rich and fundamental domains, they do not exhaust the broader landscape of reasoning abilities that a comprehensive benchmark should measure. In particular, the present tasks in PROOFGRID do not evaluate phenomena that arise in settings with quantification, variable binding, or more expressive semantic structures. An important direction for future work is to broaden the benchmark's coverage.

Finally, given the rapid advancement in AI capabilities, particularly in reasoning-focused models, the relative performance of various model families may change significantly in the not-too-distant future. In particular, the empirical rankings we report here should be understood as a snapshot tied to the models available during the time period of our investigation, rather than a stable characterization of model capabilities. This kind of temporal fragility is increasingly unavoidable in contemporary benchmark studies, especially in fast-moving areas of model development. We therefore expect that some of the comparative conclusions in this paper will need to be revisited as newer systems become available.

Despite these limitations, PROOFGRID represents a significant advancement in the evaluation of logical reasoning in AI systems, offering insights into both the impressive progress and the remaining challenges in this crucial domain.

## Acknowledgments

We thank Theo Evgeniou for the fruitful discussions that helped to spark this project and contributed to shaping its initial direction.

## Author Contributions

The first author conceived the benchmark, designed the tasks, generated the datasets, developed the prompts, led the analysis of the results, and wrote the manuscript (with the exception of Section 9 and Appendix C, which were written jointly). Both authors contributed to the experimental infrastructure. The second author led the execution of large-scale experiments across multiple models, including parallelized API-based evaluation pipelines, and contributed to debugging, validation, analysis, experimental reliability, and the provisioning of substantial computational resources. The first author additionally defined and implemented NDL and  $NDL_0$ , introduced the Epistemic Stability Index (ESI), proposed the use of IRT and Fisher information, and carried out the implementation and analysis of the 2PL models.

## Appendix A. NDL

NDL is a minimalist but rigorously defined formal language for writing natural-deduction proofs. It is:

1. expressive enough to represent standard natural-deduction proofs;
2. simple enough to admit a precise interpreter-based semantics;
3. unforgiving enough to expose reasoning errors rather than hide them behind informal prose.

Below is a sample proof that was already seen earlier, deriving the conclusion  $(B \vee D)$  from the three premises  $(A \Rightarrow B)$ ,  $(\neg A \Rightarrow C)$ , and  $(C \Rightarrow D)$ . Comments in NDL start with the hash sign #.

```

1 assert premise-1 := (A ==> B)
2 assert premise-2 := (~ A ==> C)
3 assert premise-3 := (C ==> D)
4
5 # We prove (B | D) by a case analysis on (A | ~ A), which holds by the law of the excluded middle.
6 {
7 (A | ~A) BY ex-middle on A;
8
9 # Case 1: Show that A implies (B | D)
10 assume A {
11 # Applying modus ponens to premise-1 and the assumption A gives B:
12 B BY mp on premise-1, A;
13 # So now (B | D) follows by disjunction introduction:
14 (B | D) by left-either on B, D
15 };
16
17 # Case 2: Show that (~ A) also implies (B | D)
18 assume (~ A) {
19 C BY mp on premise-2, (~ A);
20 D BY mp on premise-3, C;
21 (B | D) BY right-either on B, D
22 };
23
24 # The case analysis is now complete:
25 (B | D) BY cases on (A | ~A), (A ==> B | D), (~A ==> B | D)
26 }

```

Checking an NDL proof  $D$  amounts to evaluating  $D$  by running it through the NDL interpreter  $\mathcal{I}$ .<sup>98</sup> The evaluation of a proof can only result in one of two outcomes: either the proof works and it produces a unique *conclusion*  $p$  as its result, or the proof fails and its evaluation generates a relevant error message.

An NDL proof is always evaluated in the context of—or relative to—an *assumption base*  $\beta$ . More precisely, the NDL interpreter  $\mathcal{I}$  is a function of two arguments: a proof  $D$  and an assumption base  $\beta$ . Thus, we may only write  $\mathcal{I}(D, \beta)$ , not  $\mathcal{I}(D)$ .<sup>99</sup> An assumption base is just a finite set of formulas that are taken for granted for the purposes of a proof. Initially, the assumption base is empty:  $\beta = \emptyset$ . The keyword **assert** is then used to insert—and name—as many premises as we like into the top-level (global) assumption base. By convention, premises in PROOFGRID are always sequentially named: premise-1, premise-2, etc. These names can then be used to refer to the corresponding premises without having to write out the formulas inline. In this example we have three **assert** directives on lines 1–3, which insert the corresponding premises into the assumption base. Thus, by the time we get to line 4, the top-level  $\beta$  contains exactly 3 formulas.

Following the assertions, we have the main proof (lines 6–26). This is a *composite* proof of the form  $\{D_1; D_2; \dots; D_m\}$ , where  $D_1, \dots, D_m$  are the *subproofs* being composed. The evaluation of this proof starts on line 6, in the assumption base that contains the 3 premises that were asserted earlier. As control flows through the proof, the assumption base grows and shrinks to reflect the conclusions derived by the various rule applications, such as the application of `ex-middle` on line 7 or `mp` (modus ponens) on line 12; or the hypotheses  $p$  that are conditionally postulated when we enter a subproof of

<sup>98</sup> We use the letter  $D$  as a variable ranging over NDL proofs ( $D$  stands for “deduction,” which we use interchangeably with “proof”).

<sup>99</sup> Unless we explicitly *curry*  $\mathcal{I}$  and give it the signature:  $\mathcal{I} : \mathbf{Ded} \rightarrow \mathbf{AB} \rightarrow \mathbf{P} \cup \{\text{error}\}$ , so that it takes a proof  $D \in \mathbf{Ded}$  followed by an assumption base  $\beta \in \mathbf{AB}$  and outputs either a formula  $p \in \mathbf{P}$  or the distinguished constant *error*. In that case,  $\mathcal{I}(D)$  would be a unary function of an assumption base. Here we stick with the rectangular signature  $\mathcal{I} : \mathbf{Ded} \times \mathbf{AB} \rightarrow \mathbf{P} \cup \{\text{error}\}$ .

the form **assume**  $p$   $D'$  and then automatically discharged after  $D'$ . In this example, the first element of the composition is the application of `ex-middle` to  $A$ , which produces the conclusion  $(A \vee \neg A)$ . The proof then proceeds by a case analysis of this disjunction. It first shows, on lines 10–15, that *if* we assume  $A$ , we can derive the target conclusion  $(B \vee D)$ ; and then it shows (lines 18–22) that we can do the same if we assume the negation of  $A$ . Since we already know from line 7 that either  $A$  or its negation holds, case analysis (captured by the inference rule `cases`) allows us to infer the desired  $(B \vee D)$  (line 25).

In general, a core NDL proof  $D$  is one of three forms:

1. A *rule application*, namely, an application of an inference rule  $R$  to a number of arguments  $p_1, \dots, p_k$ , written as

$$R \text{ on } p_1, \dots, p_k \quad (\text{A17})$$

where  $R$  ranges over 24 built-in inference rules, from introduction and elimination rules for the 5 logical connectives to higher-level rules that are included for convenience, such as `dsyl` (disjunctive syllogism) and `dm` (De Morgan's).<sup>100</sup> Typically, a rule application of the form (A17) works as follows: if the arguments  $p_1, \dots, p_k$  are in the assumption base, then  $R$  derives some appropriate conclusion  $p$ , which becomes the result of (A17). But if some  $p_i$  is not in the assumption base, the application fails with an appropriate error message.

For instance, consider the application of `mp` to `premise-1` and  $A$ , on line 12. At that point in the proof, both of these arguments are in the assumption base: `premise-1` is in the assumption base because it was put there by the **assert** on line 1, while  $A$  is in the assumption base because it was postulated as a hypothesis by the **assume** on line 10, and at line 12 we are still squarely inside the scope of that hypothesis (meaning that  $A$  is still in the assumption base), as that scope does not close until line 15. Thus, because both of its arguments are in the assumption base, `mp` happily produces the conclusion  $B$ , which becomes the output or result of the step on line 12.

If one of these arguments were *not* in the assumption base, the application of `mp` would cause an error. For instance, suppose that line 12 was instead “**D BY mp on** ( $A \implies D$ ),  $A$ ”. In that case the first argument, ( $A \implies D$ ), would not be in the assumption base, and the application would be rejected as unsound.

While this is the typical mode of operation, it is not always the case that every single argument  $p_i$  in a rule application (A17) needs to be in the assumption base at the time of the application. This depends on the rule  $R$ . For example, the unary rule `ex-middle` takes an arbitrary formula  $p$  as its sole argument and always produces the conclusion  $(p \vee \neg p)$ , without caring about whether  $p$  is in the assumption base. (Intuitively, this is because the conclusion  $(p \vee \neg p)$  is always valid; whether  $p$  is in the assumption base is immaterial.)

2. A *conditional proof* of the form

$$\text{assume } p \ D', \quad (\text{A18})$$

where  $p$  is the *hypothesis* and  $D'$  is the *body* of the conditional proof, representing the *scope* of the hypothesis  $p$ . The evaluation of such a proof in a given  $\beta$  is straightforward: we add the hypothesis  $p$  to  $\beta$  and evaluate the body  $D'$  in  $\beta \cup \{p\}$ ; if and when that evaluation produces a conclusion  $q$ , we produce the conditional  $(p \implies q)$  as the result of (A18). Otherwise, if the evaluation of  $D'$  in  $\beta \cup \{p\}$  generates an error, that error is returned as the result of (A18).

3. A *composite proof* (or simply “composition”) of the form  $\{D_1; D_2\}$ , or more generally,  $\{D_1; D_2; \dots; D_m\}$  for  $m > 1$ .<sup>101</sup> Composition is the lemma-formation mechanism in NDL. To evaluate a proof of this form in a given  $\beta$ , we start out by evaluating  $D_1$  in  $\beta$ . If that yields a conclusion  $p_1$ , we proceed to evaluate  $D_2$  in  $\beta \cup \{p_1\}$ . Thus, *the conclusion of  $D_1$  is now available in  $D_2$*  (and indeed in every subsequent  $D_i$ ), essentially becoming a lemma that downstream proofs

<sup>100</sup> A complete list of NDL's inference rules and their operational semantics can be found online in the prompts for the tasks that involve NDL, such as **PL1-PW**.

<sup>101</sup> The composition operator `;` associates to the right, so  $\{D_1; D_2; \dots; D_m\}$  is syntax sugar for  $\{D_1; \{D_2; \{ \dots D_m \} \dots \}\}$ .

can use as needed. If the evaluation of  $D_2$  in  $\beta \cup \{p_1\}$  produces a conclusion  $p_2$ , we go on to evaluate  $D_3$  in  $\beta \cup \{p_1, p_2\}$ , and so on. Continuing this way, the conclusion of the rightmost subproof  $D_m$  becomes the conclusion of the entire composition. If any  $D_i$  generates an error, then the entire composition produces that error as its output.

Note that rule applications are primitive or atomic proofs, i.e., they do not contain any subproofs. Conditional proofs and compositions are complex, meaning that they (recursively) contain subproofs.

Any NDL proof  $D$  can also be written in conclusion-annotated style, as  $p \text{ BY } D$ . This can be viewed as an annotation or a promise of sorts: It says that the evaluation of proof  $D$  (in some appropriate  $\beta$ ) must produce the conclusion  $p$ . Accordingly, it is an error if we evaluate  $D$  and obtain a conclusion  $p' \neq p$ . This construct does not introduce any new inferential power, it only enforces alignment between proof intent and result.

The above was a description of both the concrete and abstract syntax of NDL, and indeed of its formal semantics as well. We essentially defined the interpreter  $\mathcal{I}$  as shown in Figure A49.

Here is the main soundness and completeness result for NDL.

**Theorem A1.** *If  $\mathcal{I}(D, \beta) = p$  then  $\beta \models p$ . Conversely, if  $\beta \models p$  then there is some NDL proof  $D$  such that  $\mathcal{I}(D, \beta) = p$ .*

The first (soundness) half says that if a proof successfully derives  $p$  from some  $\beta$ , then  $p$  follows logically from  $\beta$ . The second half (completeness) says that every logical consequence of  $\beta$  is derivable in NDL. We do not prove these results here, but briefly, soundness is shown by a straightforward structural induction, while completeness is most easily established by showing that a logic system that is already known to be complete can be embedded in NDL.

Along with a full listing of the various inference rules, this constitutes a complete description of NDL. The lone bell-and-whistle that NDL allows beyond what we have already covered is this: naming the results of intermediate subproofs and/or hypotheses. Any element  $D_j$  of a composite proof  $\{D_1; D_2; \dots; D_m\}$  can be instead written as  $I := D_j$ , and then going forward (for all subsequent deductions  $D_{j'}, j' > j$ ), the name  $I$  will be bound to the conclusion produced by  $D_j$ . Likewise, any conditional proof **assume**  $p \ D'$  can instead be written as **assume**  $I := p \ D'$ , in which case the name  $I$  will become bound to the hypothesis  $p$  inside (and only inside) the body  $D'$ . Thus, for example, we can write:

```
assume hyp := ((A & B) & (C & D)) {
```

$$\mathcal{I}(R \text{ on } p_1, \dots, p_n, \beta) = \begin{cases} q' & \text{if } R \equiv \text{mp}, n = 2, p \text{ is of the form } (q \Rightarrow q'), p_2 \equiv q, \text{ and } \{p_1, p_2\} \subseteq \beta; \\ (p_1 \& p_2) & \text{if } R \equiv \text{both}, n = 2, \text{ and } \{p_1, p_2\} \subseteq \beta; \\ q_1 & \text{if } R \equiv \text{left-and}, n = 1, p_1 \text{ is of the form } (q_1 \& q_2), \text{ and } \{p_1\} \subseteq \beta; \\ q_2 & \text{if } R \equiv \text{right-and}, n = 1, p_1 \text{ is of the form } (q_1 \& q_2), \text{ and } \{p_1\} \subseteq \beta; \\ (p \mid \sim p) & \text{if } R \equiv \text{ex-middle}, n = 1, \text{ and } p_1 \equiv p; \\ (p_1 \mid p_2) & \text{if } R \equiv \text{left-either}, n = 2, \text{ and } \{p_1\} \subseteq \beta; \\ (p_1 \mid p_2) & \text{if } R \equiv \text{right-either}, n = 2, \text{ and } \{p_2\} \subseteq \beta; \\ r & \text{if } R \equiv \text{cases}, n = 3, p_1 = (q_1 \vee q_2), p_2 = (q_1 \Rightarrow r), p_3 = (q_2 \Rightarrow r), \\ & \{p_1, p_2, p_3\} \subseteq \beta; \\ \vdots & \vdots \\ \text{error} & \text{otherwise} \end{cases}$$

$$\mathcal{I}(\text{assume } p \ D', \beta) = \mathcal{I}(D', \beta \cup \{p\})$$

$$\mathcal{I}(\{D_1; D_2\}, \beta) = \mathcal{I}(D_2, \beta \cup \{\mathcal{I}(D_1, \beta)\})$$

$$\mathcal{I}(p \text{ BY } D, \beta) = \begin{cases} p & \text{if } \mathcal{I}(D, \beta) = p \\ \text{error} & \text{otherwise} \end{cases}$$

**Figure A49.** Definition of the NDL interpreter function.

```

l := (A & B) BY left-and on hyp;
r := (C & D) BY right-and on hyp;
(r & l) BY both on r, l
}

```

and this will produce the conditional  $((A \wedge B) \wedge (C \wedge D)) \Rightarrow ((C \wedge D) \wedge (A \wedge B))$ .<sup>102</sup> Strictly speaking, this means that the interpreter  $\mathcal{I}$  takes 3 arguments: a proof  $D$ , an assumption base  $\beta$ , and a *lexical environment*  $\rho$  that maps names to formulas. For those with a background in programming language theory, extending the denotational-semantics definition of  $\mathcal{I}$  given here to account for  $\rho$  is a straightforward exercise.

For simplicity, we assumed that there is a single distinguished value *error*, but in practice the NDL interpreter produces different types of error messages. As discussed elsewhere, most errors<sup>103</sup> fall into one of the following three categories:

- *Syntax errors*: These could involve either formulas (e.g., unbalanced parentheses) or NDL-specific constructs (e.g., extra or missing curly braces, hallucinated syntax forms, and so on).
- *Structural errors*: These are caused by an incorrect application of an inference rule, e.g., to the wrong number of arguments or to arguments of the wrong form.
- *Logic errors*: These occur when a rule application appeals to a formula that does not currently hold (is not in the current assumption base).

## Appendix B. Item Response Theory

### 1. Introduction

Item Response Theory (IRT) (Hambleton et al. 1991; Reise and Revicki 2014) is a family of statistical models for developing tests and for analyzing task or benchmark data at the level of individual items/problems. Although IRT was not used in the development of PROOFGRID, we advocate for its use in analyzing and understanding various tasks in AI benchmarks, in assessing their discriminating ability, and in interpreting benchmark results. In this appendix we introduce IRT and illustrate its application to a number of PROOFGRID tasks. The theory is valuable for transparency because it can reveal both the strengths and limitations of a benchmark. In our case, IRT indicates that most PROOFGRID tasks are highly effective at discriminating mid-tier models, and several also separate top-tier models well, but none are particularly effective at distinguishing much weaker models.

Instead of summarizing performance only with aggregate accuracy, IRT models the probability that a respondent answers a particular item correctly as a function of two kinds of latent variables: properties of the respondent (typically an unobserved *ability* or proficiency, usually denoted by  $\theta$ ) and properties of the item (such as its difficulty and, in richer models like the ones we build here, its discrimination or susceptibility to guessing). In this way, IRT treats observed responses as noisy measurements generated by an underlying interaction between respondents and items, rather than as interchangeable Bernoulli trials.

We assume here that a model’s responses on a given task are driven primarily by one underlying latent variable. This is known as *unidimensionality*, and it’s a standard assumption of IRT.<sup>104</sup> This dimension is allowed to vary from task to task. Proof checking may tap a different ability than proof writing, and proof masking and gap filling may rely on other abilities still. Unidimensionality never quite holds in practice, since there are always multiple factors impinging on any test taker’s performance, from additional abilities (beyond the one that the task aims to measure) to more peripheral factors like motivation (in the case of humans) or sampling parameters like temperature (in the case of language models). The assumption can be considered sufficiently satisfied as long as there is one *dominant* factor responsible for the test taker’s performance (Hambleton and Swaminathan 2013)[p. 17]. Essentially,

<sup>102</sup> Since NDL is a proper subset of Athena, all proofs in this paper can be evaluated online at <https://tryathena.org/>. But note that Athena does not treat the names A, B, . . . , as special in any way, so to declare these as atomic sentences, the line `declare A, B, C, D, E, F, G, H: Boolean` should be prepended to your input.

<sup>103</sup> Some errors do not neatly fit into these categories (for instance, references to out-of-scope identifiers).

<sup>104</sup> Multidimensional IRT models are possible, but unidimensionality is the default in most applications.

what matters is that once we know a subject's position on the relevant latent axis, there should not be a large amount of systematic variation left over that would call for explanation in terms of additional hidden skills. In addition, task items are supposed to be conditionally independent, an assumption that can be more problematic for some AI benchmarks; we discuss this in greater detail on p. 148.

A key benefit of IRT is that it separates benchmark difficulty from model capability. Two language models can have the same overall accuracy while differing in the kinds of problems they can solve; likewise, two problems solved correctly by the same fraction of models need not be equally informative. IRT makes these distinctions explicit by estimating item parameters and placing models and items on a shared latent scale. This supports analyses such as identifying unusually easy or hard items, finding items that best distinguish between stronger and weaker models, and constructing score estimates that are less dependent on the particular mix of items in a benchmark task.

For small to moderately sized benchmark tasks, like those in PROOFGRID, all of which have fewer than 1K items, Wright maps are a particularly useful part of the IRT toolbox. A Wright map is a visualization that puts test takers and test items on the same scale. On the left, it shows where the respondents fall, from lower-performing to higher-performing. On the right, it shows where the problems fall, from easier to harder. The key idea is that both are placed on one common line, so one can immediately see whether the questions are well matched to the group being tested. If most questions sit far below the test takers, the test is too easy; if they sit far above them, it is too hard. A Wright map also shows whether the questions cover the full range of ability or bunch up in just one part of the scale. In that way, it gives a quick visual summary of how informative and well targeted a task is. We publish Wright maps for all PROOFGRID tasks for which we have built IRT models; see the last four sections of this appendix for the Wright maps of **PL1-PM**, **PL3-PC**, **PL4-PW**, and select **PL3-PW** subtasks.

In general, in the context of AI evaluation IRT provides a principled way to move from raw benchmark outcomes to a measurement model of competence. This can be useful when comparing systems across benchmark versions, studying whether some questions are redundant or poorly calibrated, or asking whether improvements reflect broad capability gains versus progress on a narrow subset of items.

## 2. Modeling and ICCs

We assume we have  $N$  respondents and  $M$  task items (problems). In this paper the respondents are language models and  $N = 24$ . Taking **PL1-PW** as an example, we would have  $M = 400$  (the number of items in **PL2-PW**). We also assume we have test data: a binary response matrix  $\mathbf{y} \in \{0, 1\}^{N \times M}$ , where  $y_{ij}$  indicates whether model  $i$  solved problem  $j$  correctly. For **PL2-PW**,  $y_{ij} = 1$  means that model  $i$  wrote a proof that derived the expected result for problem  $j$ , and  $y_{ij} = 0$  means that the model's output was incorrect.

In IRT we begin by specifying a probabilistic response model for each item  $j$ . In the 2PL model,<sup>105</sup> task item  $j$  is associated with an item response function of the following form:<sup>106</sup>

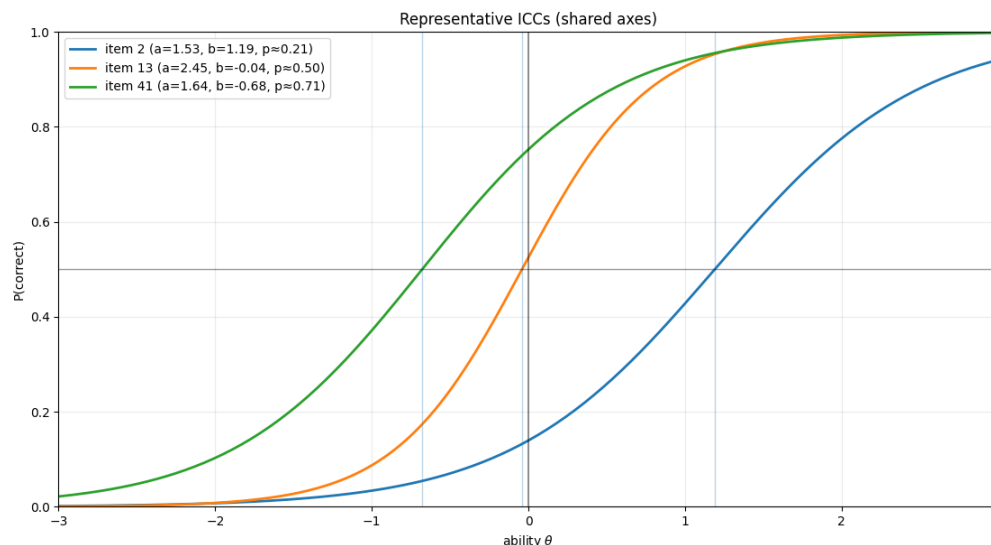
$$P(Y_j = 1 \mid \theta, a_j, b_j) = \sigma(a_j(\theta - b_j)) = \frac{1}{1 + e^{-a_j(\theta - b_j)}}, \quad (\text{A19})$$

which gives the probability that a hypothetical respondent of ability  $\theta$  answers item  $j$  correctly, given the item's key characteristics, namely its *discrimination*  $a_j$  and its *difficulty*  $b_j$ .

It is important to understand the intuition behind the algebraic form  $a_j(\theta - b_j)$ . Why that particular expression and not some other linear combination  $c_1\theta + c_2a_j + c_3b_j$ ? The first thing to note is that  $(\theta - b_j)$  represents how much the respondent's ability exceeds the item's difficulty. If  $\theta$  is greater than  $b_j$ , we should expect the probability of a correct response to be greater than a random guess, i.e. greater than  $1/2$ . If  $\theta$  is less than  $b_j$ , we should expect the probability of a correct response to be less than  $1/2$ .

<sup>105</sup> In IRT (and psychometrics in general), a "2PL model" refers to a 2-parameter logistic model.

<sup>106</sup> We write  $\sigma$  for the sigmoid function,  $\sigma(x) = 1/(1 + e^{-x})$ .



**Figure A50.** Three ICCs for three sample PL2-PW problems.

And if  $\theta$  is equal to  $b_j$ , it should be anybody's guess whether the response will be correct, i.e.,  $\theta = b_j$  should imply  $P(Y_j = 1 \mid \theta, a_j, b_j) = 0.5$ . And indeed, when  $\theta = b_j$ , the expression on the right-hand side of (A19) becomes  $1/2$ . So this is the first key idea, that the chance of success depends on the margin  $\theta - b_j$ . The second key idea is that  $a_j$  controls how sharply the probability changes as that margin changes. If the item has high discriminating power, then a small increase in ability makes a big difference in the odds of a correct response, whereas if  $a_j$  is small then the item is fuzzier and the probability of success changes much more gradually. This is what multiplying  $a_j$  by the margin ( $\theta - b_j$ ) gives us. To make this concrete, Table A33 shows how the probability of success varies for different sample values of  $\alpha$  and the margin  $\theta - b$ . In summary:

- When ability equals difficulty, the item sits at its “middle point” and the probability of success is  $1/2$ .
- Keeping the difficulty  $b_j$  constant, higher ability increases the chances of success.
- Keeping the ability  $\theta$  constant, higher difficulty decreases the chances of success.
- Discrimination controls the sensitivity of the curve, i.e., how sharply the margin matters.

We can visualize a probability model of the form (A19) by plotting the probability on the  $y$ -axis as a function of ability  $\theta$  on the  $x$ -axis. By the definition of (A19), the result will be a sigmoid curve. The item's difficulty  $b_j$  shifts the curve to the left or right, while  $a_j$  determines its slope. The resulting plot is called an *item characteristic curve* (ICC). Figure A50 overlays three sample ICCs on the same axes, for these three PL2-PW problems: items 2, 13, and 41. Again, the horizontal axis is model ability  $\theta$  and the vertical axis is the probability of answering an item correctly. Each S-shaped curve corresponds to one benchmark item. Where a curve crosses the 50% line marks that item's difficulty  $b$ : curves shifted to the right (like item 2,  $b \approx 1.19$ ), are harder—you need higher  $\theta$  to reach a 50% success chance. Curves

$\alpha \setminus (\theta - b_j)$	-3	-2	-1	-0.5	0	0.5	1	2	3
0.2	0.354	0.401	0.450	0.475	0.500	0.525	0.550	0.599	0.646
0.5	0.182	0.269	0.378	0.438	0.500	0.562	0.622	0.731	0.818
1	0.047	0.119	0.269	0.378	0.500	0.622	0.731	0.881	0.953
2	0.002	0.018	0.119	0.269	0.500	0.731	0.881	0.982	0.998
3	0.000	0.002	0.047	0.182	0.500	0.818	0.953	0.998	1.0

**Table A33.** Success probabilities under the 2PL model,  $p = \sigma(\alpha(\theta - b_j))$ , for several values of the discrimination parameter  $\alpha$  and the margin  $\theta - b_j$ . As the margin increases, the probability of success rises; larger values of  $\alpha$  make this rise steeper.

shifted to the left (like item 41,  $b \approx -0.68$ ) are easier, with even lower-ability models having a good chance at writing a correct proof. The thin vertical lines (in light blue color) at each  $b$  visualize this “where  $P = 0.5$ ” point, and the darker vertical line at  $\theta = 0$  plus the horizontal line at  $P = 0.5$  help to identify those crossings.

The steepness of a curve around its midpoint reflects the item’s discrimination  $\alpha$ —how sharply it separates nearby ability levels. A steeper curve (e.g., item 13 with  $\alpha \approx 2.45$ ) makes a big probability jump with a small change in  $\theta$ , so it’s excellent for telling slightly weaker from slightly stronger models near its difficulty. A flatter curve changes more gradually, so it’s less informative for fine-grained ranking. In short, horizontal position reflects difficulty ( $b$ ), while slope reflects discrimination ( $a$ ). Looking at all three together shows how this task mixes easier and harder questions, and which ones are better at distinguishing models that are close in overall skill.

We can estimate the  $N + 2 \cdot M$  parameters  $\theta_i, a_j, b_j$  (the ability of each respondent plus each item’s discrimination and difficulty) by maximizing the likelihood of the observed data  $\mathbf{y}$ :

$$\begin{aligned} (\{\hat{\theta}_i\}, \{\hat{a}_j\}, \{\hat{b}_j\}) &= \operatorname{argmax}_{\{\theta_i\}, \{a_j\}, \{b_j\}} \prod_{i=1}^N \prod_{j=1}^M P(Y_{ij} = y_{ij} \mid \theta_i, a_j, b_j) \\ &= \operatorname{argmax}_{\{\theta_i\}, \{a_j\}, \{b_j\}} \prod_{i=1}^N \prod_{j=1}^M \left[ P(Y_{ij} = 1 \mid \theta_i, a_j, b_j)^{y_{ij}} \cdot \left[ 1 - P(Y_{ij} = 1 \mid \theta_i, a_j, b_j) \right]^{1-y_{ij}} \right] \\ &= \operatorname{argmax}_{\{\theta_i\}, \{a_j\}, \{b_j\}} \prod_{i=1}^N \prod_{j=1}^M \left[ \frac{1}{1 + e^{-a_j(\theta_i - b_j)}} \right]^{y_{ij}} \cdot \left[ 1 - \left[ \frac{1}{1 + e^{-a_j(\theta_i - b_j)}} \right] \right]^{1-y_{ij}} \\ &= \operatorname{argmax}_{\{\theta_i\}, \{a_j\}, \{b_j\}} \log \prod_{i=1}^N \prod_{j=1}^M \left[ \frac{1}{1 + e^{-a_j(\theta_i - b_j)}} \right]^{y_{ij}} \cdot \left[ 1 - \left[ \frac{1}{1 + e^{-a_j(\theta_i - b_j)}} \right] \right]^{1-y_{ij}} \\ &= \operatorname{argmax}_{\{\theta_i\}, \{a_j\}, \{b_j\}} \sum_{i=1}^N \sum_{j=1}^M y_{ij} \cdot \log \left[ \frac{1}{1 + e^{-a_j(\theta_i - b_j)}} \right] + (1 - y_{ij}) \cdot \log \left[ 1 - \left[ \frac{1}{1 + e^{-a_j(\theta_i - b_j)}} \right] \right] \end{aligned}$$

where we write  $Y_{ij}$  to denote the random variable that records whether respondent  $i$  answers item  $j$  correctly, with  $Y_{ij} = 1$  for a correct response and  $Y_{ij} = 0$  for an incorrect one. We fix the latent scale of  $\theta$  by requiring  $\theta \sim \mathcal{N}(0, 1)$ , i.e., that  $\theta$  is normally distributed with a 0 mean and standard deviation 1; and we also impose  $a_j > 0$  for all  $j = 1, \dots, M$ . We describe our algorithm for estimating these parameters in Section 4. (Note that we introduce estimation here via the unregularized joint likelihood for conceptual clarity. In practice, as described in Section 4, the models reported in this appendix are fit by joint MAP estimation with Gaussian priors, which regularizes the item and respondent parameters and improves stability when the number of items is large relative to the number of respondents.)

Once these parameters are estimated, we have built the desired model (A19): we can plug in an arbitrary  $\theta$  and compute the associated probability. The main use of such a model, however, is not prediction (as is usually the case in ML applications), but rather explanation (e.g., structured explanations of which items are hard/easy, which discriminate, where each model lies on the ability scale, and so on).

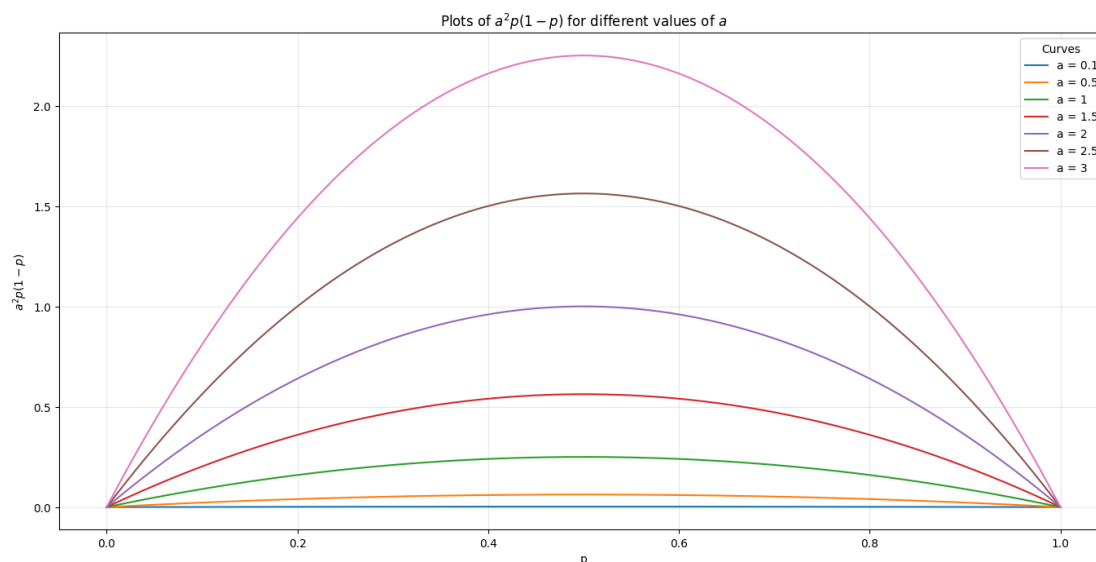
Note that the product on the first identity of the above chain is justified only under an independence assumption: once we condition on the latent trait, item responses should be independent. Informally, this means that the only reason why two items might co-vary is that stronger respondents do better on both. This is a fundamental assumption of IRT. Violations are common when items are paraphrases (or duplicates), share context, depend on the same passage, or reward the same narrow trick. One might think that this is not a concern for stateless language models, but the issue is not about inter-item memory. The issue is that even after conditioning on latent ability, success on item  $j_1$  might still tell us something about success on  $j_2$ , because there may be additional shared factor(s) on which both items depend. That said, if interpreted strictly the condition would be impossible to

satisfy,<sup>107</sup> so in practice it is usually understood as an idealization and its satisfaction is considered a matter of degree.

Some tasks in PROOFGRID appear to be more at odds with the fundamental assumptions of IRT. **PL3-PW** performance, in particular, has a nontrivial dependence on problem type. This threatens conditional independence because many items from the same problem family are not just “another draw of the same skill.” They often share the same hidden structure, so a model that has the right trick for one item will tend to do well on related items, and a model that lacks that trick will tend to fail on many of them. Unidimensionality is also suspect here, since different problem families seem to demand different types of competence (the ability to perform simple forward inference vs counting arguments etc.). There may still be one *dominant* latent factor at play, namely proof-writing ability, but it is an empirical question whether that factor is strong enough to explain most of the covariance among item responses. In **PL3-PC**, by contrast, problem families play a much less prominent causal role. The model there is given a list of mini-SAT steps to verify. As long as these steps are roughly of the same size and structure across two problem families, then the identities of these families should not matter much. Proof length and formula size are more important factors (and explain why De Bruijn proofs, for example, as so hard to verify). We build IRT models for most PROOFGRID tasks, except that in **PL3-PW** we build different models for different problem families. In this appendix we present IRT models for **PL1-PM**, **PL3-PC**, **PL4-PW**, and subsets of **PL3-PW** (see Section 5—Section 8).

### 3. Introducing Normalized Discrimination Scores

Our objective in this section is to introduce and motivate a single quantity that measures how strongly discriminating a task is in a given band of ability. The quantity should be normalized to the unit interval, with values close to 0 indicating very poor discriminating power and values close to 1 indicating very high discriminating power. We base our development on the notion of Fisher information (Ly et al. 2017), which we discuss below in detail.



**Figure A51.** Fisher information of a task item as a function of different  $a$  (discrimination) values.

<sup>107</sup> Consider two very different items on a math test. While they may be testing different math skills and content, both may depend on the ability to compute binomial roots or apply the Pythagorean theorem, both of which have pervasive uses in mathematics.

For an item  $j$  with discrimination  $a_j > 0$  and difficulty  $b_j \in \mathbb{R}$ , equation (A19) specifies the probability that a language model of latent ability  $\theta$  solves the item as

$$P_j(\theta) = \sigma(a_j(\theta - b_j)) = \frac{1}{1 + e^{-a_j(\theta - b_j)}}. \quad (\text{A20})$$

The Fisher item information at ability  $\theta$  is defined as:

$$I_j(\theta) = a_j^2 P_j(\theta) (1 - P_j(\theta)). \quad (\text{A21})$$

To understand the intuition behind this quantity, consider the algebraic behavior of the expression  $a_j^2 P_j(\theta) (1 - P_j(\theta))$ . First, the bigger the item's discrimination,  $a_j$ , the bigger the  $I_j(\theta)$ , i.e., the higher the information conveyed by the item, and the growth is quadratic: doubling  $a_j$  quadruples  $I_j(\theta)$ , so this is a strong effect. The second and more important term is the product  $P_j(\theta) (1 - P_j(\theta))$  (which, as we note below, is really the variance of a Bernoulli random variable). When  $P_j(\theta)$  is low, i.e., when a model of ability  $\theta$  is unlikely to solve the item, this product is low. Likewise, the product is low when  $P_j(\theta)$  is high. The product is maximized at 0.5. When we fix a value for  $a$ , the quantity  $a^2 p (1 - p)$  can be plotted as a function of  $p$ , and Figure A51 does this for a range of  $a$  values. We thus see that:

- if the item is very likely to be failed by a model of ability  $\theta$ , the information is low;
- if the item is very likely to be solved by a model of ability  $\theta$ , the information is also low; and
- the information is highest when the response is most uncertain, i.e., when the response could be correct or incorrect with roughly equal probability.

This holds for any value of  $a$ , as shown in Figure A51, although higher  $a$  values result in much higher information for any fixed  $P_j(\theta)$ .

For a more statistical perspective: If we define  $X_j^\theta$  as a Bernoulli variable representing a hypothetical binary response to item  $j$  from a respondent of ability  $\theta$ , then the success-probability parameter of that variable is precisely  $P_j(\theta)$ , i.e.,  $X_j^\theta \sim \text{Bernoulli}(P_j(\theta))$ , and its variance is  $P_j(\theta) \cdot (1 - P_j(\theta))$ . When  $P_j(\theta)$  is low or high, that variance is small because most responses will be either incorrect or correct, respectively. The variance increases only when the probability enters middle-ground territory, in which case we can expect more diversity in the correctness of the responses. So the variance term  $P_j(\theta) \cdot (1 - P_j(\theta))$  measures the amount of uncertainty regarding responses to item  $j$  from test takers of ability  $\theta$ , and Fisher information is exactly this response variability weighted by the square of the item's discrimination ( $a_j$ ). For a more analytical perspective, observe that the slope of the ICC of item  $j$  at a given  $\theta$  is the derivative of (A20) with respect to  $\theta$ , which is precisely  $a_j P_j(\theta) (1 - P_j(\theta))$ . Therefore, in a 2PL model like this one,  $I_j(\theta) = a_j P_j'(\theta)$  and item information is proportional to the local slope of the ICC, with  $a_j$  as the proportionality factor.

It's worth stressing that this information quantity is not an intrinsic property of an item  $j$ ; it depends crucially on the given  $\theta$ . One and the same item could be highly discriminating (informative) for one ability level and quite ineffective for another ability level. As a concrete example, consider item 2 in **PL2-PW**, with  $a_2 = 1.53$  and  $b_2 = 1.19$ . For  $\theta = 1.2$  (roughly the ability of Gemini 3), we have  $P_2(1.2) = 0.504$  and thus  $1 - P_2(1.2) = 0.496$ , and hence

$$I_2(\theta = 1.2) = a_2^2 P_2(1.2)(1 - P_2(1.2)) = 2.3409 \cdot 0.504 \cdot 0.496 = 0.58.$$

However, the information of the same item at the level of  $\theta = -1.0$ , roughly the ability of Mistral Magistral and Grok-3, is

$$I_2(-1.0) = a_2^2 P_2(-1.0)(1 - P_2(-1.0)) = 2.3409 \cdot 0.034 \cdot 0.966 = 0.07.$$

This represents an 88% relative drop in information content between these two ability levels.

In the 2PL models that we use here, the Fisher information  $I_j(\theta)$  of item  $j$  is always maximized at  $\theta = b_j$ , since that is precisely the point where  $P_j(\theta) = 0.5$  and  $a_j^2 P_j(\theta) (1 - P_j(\theta))$  attains the largest possible value.

Next, given a task  $T$ , we define the *task information* of  $T$  at an ability  $\theta$  as the sum of  $I_j(\theta)$  across all items  $j$  in  $T$ :

$$I_T(\theta) = \sum_{j=1}^{|T|} I_j(\theta), \quad (\text{A22})$$

where  $|T|$  is the number of items in task  $T$  (only counting items contributing to the fit after filtering uninformative all-correct/all-incorrect columns).

In the general theory of Fisher information, the quantity represents how much a random observation  $X$  tells us about an unknown parameter  $\theta$ , given a probability model  $f(x | \theta)$ . Its units are  $1/(\text{units of } \theta^2)$ , so if  $\theta$  is measured in meters, for example, Fisher information would be measured in  $1/m^2$  units. In the current IRT context, however,  $\theta$  is a latent standardized ability scale, not a physical quantity with real-world units. As a result, item information often feels unitless, and raw numbers like 0.4, 1.2 or 2.8 are hard to interpret. The rest of this section aims to turn this scale-dependent quantity that is hard to interpret in isolation into a  $[0,1]$ -normalized quantity representing a relative efficiency score against a clearly defined upper bound.

### Restricting to a target ability band.

As defined by (A22), the Fisher information of a task  $T$  is local in  $\theta$ :  $I_T(\theta)$  tells us how discriminating the task is at one specific ability level, not overall. In practice, we typically care about an entire ability region (say mid-tier models or elite models), i.e., an interval  $\Theta = [\theta_1, \theta_2]$ . Therefore, we need to turn the whole curve  $I_T(\theta)$  between  $\theta_1$  and  $\theta_2$  into a single summary number for that region. The natural way to do that is to average the task information over the chosen band. Conceptually, this is done with an integral, while numerically we implement the averaging over a fine uniform grid  $\{\theta^{(i)}\}_{i=1}^G \subset \Theta$ :

$$\bar{I}_T(\Theta) = \frac{1}{|\Theta|} \int_{\theta_1}^{\theta_2} I_T(\theta) d\theta \approx \frac{1}{G} \sum_{i=1}^G I_T(\theta^{(i)}), \quad (\text{A23})$$

where  $\theta^{(i)}$  denotes the  $i^{\text{th}}$  point in a numerical grid of size  $G$  laid across  $\Theta$ . Thus,  $\bar{I}_T(\Theta)$  is just the average height of the task-information curve over the interval of interest. If the curve is high throughout the band, the task discriminates well there. If it is high only in a narrow spike but low elsewhere, the average will reflect that. So this is just getting a band-specific single-number summary out of a location-dependent quantity, thereby answering the following question: "On average, how much local discriminating power does this task have over the ability range we care about?" Intuitively, if many items  $j$  have their ICC "humps"  $b_j$  inside  $\Theta$  or at least near it, then  $\bar{I}_T(\Theta)$  will be high, since these are the points that maximize  $I_j$ . But  $\bar{I}_T(\Theta)$  will be low if many items have their information concentrated far from  $\Theta$ . In that case there will be slim pickings for the integral (A23) to sum over.

### Mean information per item.

To make the magnitude comparable across tasks of different sizes, we define the *mean information per item over*  $\Theta$  of a task  $T$  by dividing by the number of items in  $T$ :

$$MI_T(\Theta) = \frac{\bar{I}_T(\Theta)}{|T|}.$$

This removes the trivial advantage of longer tasks (otherwise, a 400-item task will usually have much larger total information than a 50-item task, even if its items are not individually better).

This quantity does not solve the deeper interpretability problem. A value like 0.8 for  $MI_T(\Theta)$  still lacks a clean meaning. It simply gives us a size-adjusted quantity, not yet a naturally interpretable one. By itself, it is mainly useful for comparisons. If, for a given band  $\Theta$ , this quantity is 1.2 for task  $T_1$

and 0.8 for task  $T_2$ , we can say that task  $T_1$  has more average discrimination per item in  $\Theta$  than task  $T_2$ . This is a valid comparative statement, even if the raw numbers have no intuitive absolute benchmark.

### Normalized band-targeting score in $[0, 1]$

As already mentioned, raw Fisher information is useful but its scale is hard to interpret directly. To get a clearer score, we compare the task's band-averaged information  $\bar{I}_T(\Theta)$  to an idealized upper bound defined over  $\Theta$ .

We define this idealized upper bound as follows. For each item  $j$ , we keep its estimated discrimination  $a_j$  fixed and ask for the largest band-averaged information that could be obtained by an imaginary item with that same discrimination strength, but with difficulty chosen to best match the target band. This gives:<sup>108</sup>

$$\bar{I}_j^*(\Theta) = \max_{b \in \mathbb{R}} \frac{1}{|\Theta|} \int_{\theta_1}^{\theta_2} I_j(\theta) d\theta = \max_{b \in \mathbb{R}} \frac{1}{|\Theta|} \int_{\theta_1}^{\theta_2} a_j^2 \sigma(a_j(\theta - b)) (1 - \sigma(a_j(\theta - b))) d\theta. \quad (\text{A24})$$

Summing over items gives the band's attainable upper bound:

$$UB_T(\Theta) = \sum_{j=1}^{|T|} \bar{I}_j^*(\Theta).$$

We finally define the normalized discrimination score over a band  $\Theta$  as follows:

$$\mathcal{S}_T(\Theta) = \frac{\bar{I}_T(\Theta)}{UB_T(\Theta)} \in [0, 1]. \quad (\text{A25})$$

This can be read as follows: among all same-length comparison tasks with the same per-item discrimination profile  $(a_1, \dots, a_{|T|})$ , but with item difficulties chosen to fit this band perfectly, what fraction of the maximum attainable band-specific information is the actual task  $T$  achieving?

What does "fitting this band perfectly" mean? It means that the difficulty  $b_j$  of each item  $j$  sits right at the midpoint of  $\Theta$ . (As explained in footnote 108, the maximum value specified by A24 is always attained exactly at  $(\theta_1 + \theta_2)/2$ .) So the normalized band-targeting score answers the following question: how informative is  $T$  on  $\Theta$  compared with the unique hypothetical task  $T'$  obtained from  $T$  by keeping each item's sharpness  $a_j$  fixed but setting each item's difficulty  $b_j$  exactly to the midpoint of  $\Theta$ ? Task  $T'$  is a mathematical fiction, of course: every single item in it has the exact same difficulty  $b_j$ , namely, the midpoint of  $\Theta$ , so all ICC curves are centered at the exact same place and the only thing distinguishing them is their sharpness (the  $a_j$  values). This is highly implausible, but  $UB_T(\Theta)$  is not trying to describe a realistic benchmark design. It is a deliberately artificial upper bound for one specific band. If the only question is "how much information could these same items deliver on this one target band, if one were allowed to place their difficulties optimally for that band?", then collapsing all  $b_j$  values to the midpoint is exactly the maximizing move (in the 2PL setting).

$\mathcal{S}_T(\Theta)$  values close to 1 mean that  $T$  is very well targeted to the ability band represented by  $\Theta$ . A lower value suggests that, even if the items in  $T$  are reasonably sharp, their difficulties are not as well aligned to  $\Theta$  as they could be in theory. However, this should not be over-interpreted:  $\mathcal{S}_T(\Theta)$  is a

<sup>108</sup> The maximization operation in (A24) ranges over all  $b \in \mathbb{R}$ , so one may wonder if it is well-founded. If we define  $F_j(b, \Theta)$  as  $(1/|\Theta|) \int_{\theta_1}^{\theta_2} a_j^2 \sigma(a_j(\theta - b))(1 - \sigma(a_j(\theta - b))) d\theta$ , then the right-hand side of (A24) is maximizing  $F_j(b, \Theta)$  over all  $b \in \mathbb{R}$ . In fact, for any finite  $\theta_2 > \theta_1$  and fixed  $a_j > 0$ , the integral simplifies to a unary function of  $b$ , namely  $g(b) = a_j[\sigma(a_j(\theta_2 - b)) - \sigma(a_j(\theta_1 - b))]$  (this follows from elementary calculus, given that  $\frac{d}{db} \sigma(a_j(\theta - b)) = a_j \sigma(a_j(\theta - b))(1 - \sigma(a_j(\theta - b)))$ ), so that  $F_j(b, \Theta) = (1/|\Theta|)g(b)$ . We first note that this quantity is bounded from above, since for all  $x \in \mathbb{R}$  we have  $0 \leq \sigma(x)(1 - \sigma(x)) \leq 1/4$ , and hence for every  $b \in \mathbb{R}$ ,  $0 \leq F_j(b, \Theta) \leq a_j^2/4$ , where  $a_j$  is fixed. So this optimization operation has a finite hard ceiling. The other question is whether the maximum is actually attained at some finite  $b$ , and a simple continuity argument will show that it is. Essentially:  $F_j(b, \Theta)$  is a continuous function of  $b$ , it is bounded (as we just showed), and it vanishes at both tails (because, as  $b$  tends to either  $-\infty$  or  $+\infty$ , the quantity  $F_j(b, \Theta)$  tends to 0). This shows that the maximization operation is well defined (and a little more algebra will show that, assuming  $\theta_1 < \theta_2$ , a maximum value must occur at the midpoint  $(\theta_1 + \theta_2)/2$ ).

fraction whose denominator sets an exceedingly high bar (a very stringent idealized upper bound), so relatively moderate values may already indicate strong targeting for practical purposes.

Finally,  $S_T(\Theta)$  clearly depends on  $\Theta$ , so it is not a generic, band-independent measure of task informativeness. Moreover, its interpretation is most useful for bands of substantive interest. In one direction, if  $\Theta$  is very wide then high values become easier to obtain almost by construction. For example, if  $\Theta$  is wide enough to cover essentially the entire ability range for which the task was designed, then  $S_T(\Theta)$  will often be high because many items already have their difficulty levels somewhere inside that broad band, so moving them all to the midpoint does not improve things much. At the other extreme, if  $\Theta$  is very narrow then a high  $S_T(\Theta)$  is generally unrealistic, unless the task is unusually concentrated on that narrow slice of ability. In that case the score is comparing the real task to an idealized version in which every item is perfectly centered on that tiny interval, which is a very demanding standard and one that a broadly useful task will usually not meet. The measure is most useful for substantively meaningful ability ranges, such as mid-tier, top-tier, or lower-tier models, or more systematically defined bands such as quartiles or quantiles.

#### 4. Fitting a 2PL IRT Model by Alternating MAP–MLE (EM–style) Updates

In standard psychometric uses of IRT, the number of respondents  $N$  is typically much larger than the number of test items  $M$ . In many recent AI benchmarks, including PROOFGRID, the situation is reversed, as we often have  $M \gg N$ . This can introduce certain technical complications, because we have few data points from which to infer parameter values; with  $N = 24$  in a PROOFGRID task with  $M$  items, we have to estimate 2 parameters ( $a_j$  and  $b_j$ ) for each  $j = 1, \dots, M$  from only 24 binary responses. In this setting, item parameters can become noisy (discrimination parameters, in particular, can be especially fragile). In addition, estimation in this setting can be subject to separation pathologies and overfitting.

Here we mitigate the data asymmetry by fitting 2PL models not by gradient ascent but by using an alternating MAP/Newton procedure that is well suited to the regime of many items and relatively few respondents: Gaussian priors regularize the respondent and item parameters, optimization in  $\log a_j$  enforces positive discriminations and mitigates separation pathologies, and blockwise damped Newton updates yield stable low-dimensional subproblems for items and respondents. This doesn't quite eliminate the fundamental data asymmetry, but it makes estimation substantially more stable and interpretable than unregularized joint maximum likelihood in the same regime. The remainder of this section describes our fitting algorithm in greater detail.

#### Goal and data.

We observe a binary response matrix  $Y \in \{0, 1\}^{N \times M}$ , where  $Y_{ij} = 1$  if model (respondent)  $i \in \{1, \dots, N\}$  solves item  $j \in \{1, \dots, M\}$  and 0 otherwise. The 2-parameter logistic (2PL) model posits for each item  $j$  a discrimination  $a_j > 0$  and a difficulty  $b_j \in \mathbb{R}$ , and for each model  $i$  a latent ability  $\theta_i \in \mathbb{R}$ ,

$$P(Y_{ij} = 1 \mid \theta_i, a_j, b_j) = \sigma(a_j(\theta_i - b_j)) \quad \text{with} \quad \sigma(z) = \frac{1}{1 + e^{-z}}.$$

#### Objective.

The algorithm maximizes a regularized (MAP) joint log-posterior over all parameters,

$$\mathcal{L}(\theta, a, b) = \sum_{i=1}^N \sum_{j=1}^M \left[ Y_{ij} \log P_{ij} + (1 - Y_{ij}) \log(1 - P_{ij}) \right] - \frac{1}{2} \sum_{i=1}^N \theta_i^2 - \frac{1}{2} \sum_{j=1}^M \frac{(\log a_j - \mu_a)^2}{\sigma_a^2} - \frac{1}{2} \sum_{j=1}^M \frac{b_j^2}{\sigma_b^2},$$

where  $P_{ij} = \sigma(a_j(\theta_i - b_j))$ ,  $\mu_a = 0$ , and  $\sigma_a, \sigma_b > 0$  are prior scales. The Gaussian priors  $\theta_i \sim \mathcal{N}(0, 1)$ ,  $\log a_j \sim \mathcal{N}(\mu_a, \sigma_a^2)$ ,  $b_j \sim \mathcal{N}(0, \sigma_b^2)$  stabilize estimation (preventing infinite slopes from separation) and enforce  $a_j > 0$  by optimizing in  $\log a_j$ .

### Why an EM-style algorithm?

The likelihood couples language models and items:  $P_{ij}$  depends on both  $\theta_i$  and  $(a_j, b_j)$ . Direct Newton on *all* parameters is brittle on sparse/balanced items and under separation. Instead, we use an alternating conditional maximization strategy (a close cousin of ECM/EM): holding  $\theta$  fixed we maximize over  $(\log a, b)$  item-by-item; then holding  $(\log a, b)$  fixed we maximize over  $\theta$  model-by-model. Each conditional subproblem is a tiny, well-behaved logistic regression, so Newton steps are reliable; the alternation is a coordinate-ascent on  $\mathcal{L}$ .

### Preprocessing and initialization.

We drop non-informative columns (items with all 0 or all 1). We initialize  $\theta_i$  from smoothed logits of per-model accuracy and standardize to mean 0, variance 1. We draw  $\log a_j \sim \mathcal{N}(0, 0.25^2)$  and  $b_j \sim \mathcal{N}(0, 1)$ . We also enforce a practical box constraint  $a_j \in [a_{\min}, a_{\max}]$  after each update.

### Item (M-step for items) updates.

Fix  $\theta$ . For each item  $j$  we maximize the *itemwise* objective

$$\ell_j(\log a_j, b_j) = \sum_{i=1}^N \left[ Y_{ij} \log P_{ij} + (1 - Y_{ij}) \log(1 - P_{ij}) \right] - \frac{(\log a_j - \mu_a)^2}{2\sigma_a^2} - \frac{b_j^2}{2\sigma_b^2},$$

where now  $P_{ij} = \sigma(e^{\log a_j}(\theta_i - b_j))$ . We use a single (damped) Newton–Raphson step on the  $2 \times 2$  Hessian for  $(\log a_j, b_j)$ , with gradients/Hessians (letting  $a_j = e^{\log a_j}$ ,  $t_{ij} = \theta_i - b_j$ ,  $p_{ij} = P_{ij}$ ,  $w_{ij} = p_{ij}(1 - p_{ij})$ ):

$$\begin{aligned} \frac{\partial \ell_j}{\partial \log a_j} &= \sum_i (Y_{ij} - p_{ij}) a_j t_{ij} - \frac{\log a_j - \mu_a}{\sigma_a^2}, \\ \frac{\partial \ell_j}{\partial b_j} &= -\sum_i (Y_{ij} - p_{ij}) a_j - \frac{b_j}{\sigma_b^2}, \\ \frac{\partial^2 \ell_j}{\partial (\log a_j)^2} &= -\sum_i w_{ij} (a_j t_{ij})^2 - \frac{1}{\sigma_a^2}, \\ \frac{\partial^2 \ell_j}{\partial b_j^2} &= -\sum_i w_{ij} a_j^2 - \frac{1}{\sigma_b^2}, \\ \frac{\partial^2 \ell_j}{\partial \log a_j \partial b_j} &= -\sum_i w_{ij} (a_j t_{ij}) (-a_j). \end{aligned}$$

We solve the  $2 \times 2$  linear system for the Newton step, damp it by a factor  $\in (0, 1]$ , then clip  $a_j$  into  $[a_{\min}, a_{\max}]$  (by re-adjusting  $\log a_j$ ). This is done independently for each item  $j$ .

### Model (M-step for models) updates.

Fix  $(\log a, b)$ . For each language model  $i$  we maximize

$$\ell_i(\theta_i) = \sum_{j=1}^M \left[ Y_{ij} \log P_{ij} + (1 - Y_{ij}) \log(1 - P_{ij}) \right] - \frac{1}{2} \theta_i^2, \quad P_{ij} = \sigma(a_j(\theta_i - b_j)),$$

by one (damped) Newton step using

$$\frac{d\ell_i}{d\theta_i} = \sum_j (Y_{ij} - P_{ij}) a_j - \theta_i, \quad \frac{d^2\ell_i}{d\theta_i^2} = -\sum_j a_j^2 P_{ij}(1 - P_{ij}) - 1.$$

Again, the update is scalar per model and therefore stable.

**Identifiability rescaling.**

The 2PL model is not identifiable up to linear transforms of the latent scale:  $(\theta, a, b) \mapsto ((\theta - \mu)/s, sa, (b - \mu)/s)$  yields the same probabilities. After each iteration we fix the scale by standardizing  $\theta$  to mean 0 and variance 1:

$$\theta_i \leftarrow \frac{\theta_i - \bar{\theta}}{s}, \quad b_j \leftarrow \frac{b_j - \bar{\theta}}{s}, \quad \log a_j \leftarrow \log a_j + \log s,$$

where  $\bar{\theta}$  and  $s$  are the current mean and standard deviation of  $\theta$ . (This keeps  $a_j(\theta_i - b_j)$  invariant, hence leaves  $P_{ij}$  unchanged.)

**Monotonicity, damping, and clipping.**

The Newton steps are *damped* to avoid overshooting on items with near-separation, and  $a_j$  is clipped to  $[a_{\min}, a_{\max}]$  to prevent extreme slopes. The Gaussian priors play the role of ridge regularizers on  $\theta$ ,  $\log a$ , and  $b$ , ensuring negative-definite Hessians and preventing degeneracy.

**Convergence check.**

We track the relative change of the stacked parameter vector  $(\theta^\top, \log a^\top, b^\top)^\top$  and stop when the  $\ell_2$  change falls below a tolerance. Because each block step increases a local quadratic approximation of  $\mathcal{L}$ , the procedure behaves like a block-Newton EM/ECM and converges rapidly in practice.

**Relation to EM / MMLE.**

Classical marginal-maximum-likelihood (MMLE) IRT treats  $\theta$  as latent *random* effects, integrates them out in an E-step (via quadrature), and maximizes the marginal likelihood over  $(a, b)$  in an M-step. Our implementation instead performs *joint* MAP estimation (sometimes called MAP-JML): we maximize over both  $\theta$  and  $(a, b)$ , alternating between them. This is an EM-style coordinate-ascent on the posterior, with the Gaussian priors acting like the penalty that stabilizes each block. Both approaches are standard; MAP-JML is simple, fast, and well-suited when  $M$  is large and  $N$  is modest.

**Algorithm summary**

1. Build  $Y$  from the input dictionary; optionally drop all-0/all-1 items.
2. Initialize  $\theta$  from logits of accuracy; initialize  $\log a, b$  from Gaussians.
3. Repeat until convergence:
  - (a) (*Item step*) For each  $j$ : take one damped Newton step on  $(\log a_j, b_j)$  for  $\ell_j$ , then clip  $a_j$ .
  - (b) (*Language model step*) For each  $i$ : take one damped Newton step on  $\theta_i$  for  $\ell_i$ .
  - (c) (*Rescale*) Standardize  $\theta$  to mean 0, var 1 and adjust  $(\log a, b)$ .
4. Return dictionaries for  $\theta, a$ , and  $b$  (in original item indices).

**Why this works well in practice.**

Each subproblem is low-dimensional with closed-form gradients/Hessians, so one Newton step per block already yields large ascent. Regularization tames separation and keeps slopes finite; re-centering fixes the latent scale; and clipping prevents pathological updates. Empirically, with hundreds of items and a few dozen models, this EM-style alternating MAP procedure converges quickly and produces stable, interpretable estimates of abilities  $\theta_i$ , item difficulties  $b_j$ , and discriminations  $a_j$  suitable for our downstream analyses (ICCs, Wright maps, information curves, and the normalized discrimination scores introduced in Section 3).

**5. PL1-PM IRT model**

The ability values estimated from the PL1-PM data are shown in Table A34. They induce the same ranking as the accuracies in Table 7, except that GPT-5 and Claude Sonnet 4.5 with extended

Model	Ability $\theta$	Model	Ability $\theta$
gemini-3.1-pro-preview	2.141	gemini-3.0-pro-preview	1.429
gpt-5.2	1.114	claude-opus-4-5-20251101	1.035
gpt-5.1	1.004	grok-4	0.756
gemini-2.5-pro	0.709	claude-sonnet-4.5-ext-thinking	0.624
gpt-5	0.613	o3	0.506
deepseek-reasoner	0.176	deepseek-r1-0528	0.076
o4-mini	-0.033	claude-opus-4-20250514	-0.224
o3-mini	-0.434	claude-sonnet-4-20250514	-0.442
deepseek-v3.1	-0.561	grok-3	-0.796
deepseek-v3	-0.889	hermes-4	-0.985
llama-3.1-405b	-1.052	gpt-4.1	-1.350
mistral-magistral	-1.637	gpt-4o	-1.780

**Table A34.** PL1-PM  $\theta$  parameters estimated from the task results.

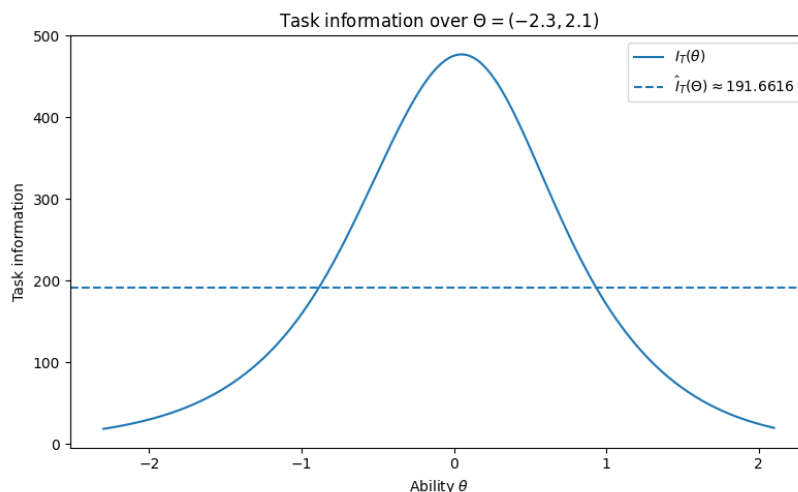
thinking are now strictly ordered even though their accuracies are identical (the latent ability of GPT-5 is estimated to be slightly lower).

The Wright map for the proof-masking task is shown below, while the task-information curve over the observed ability range appears in Figure A52.

```

PL1-PM MODEL-ITEM MAP: models and items on the same θ scale
 θ range: [-2.32, 2.14]
 θ | Models | Items (b in bin)
-----|-----|-----
-2.20 | | 41 73 93 107 140 223 243 258
-1.98 | | 119 232
-1.76 | gpt-4o | 18 166
-1.54 | mistral-magistral | 164 286
-1.31 | gpt-4.1 | 70 82 89 163 219
-1.09 | hermes4, llama_400b_nous | 8 29 30 112 129 133 135 174 193 197 221 254
-0.87 | deepseek3, grok-3 | 48 51 56 58 63 78 98 114 137 168 203 230 244
-0.64 | deepseek3.1 | 22 34 88 92 94 118 123 134 144 157 158 192 228 267 281
-0.42 | o3-mini, |
 claude-sonnet-4-20250514 | 1 19 20 27 37 38 40 44 45 52 54 55 60 65 81 86 101 108 117 121
 | 127 136 148 149 150 156 160 161 170 182 185 190 207 225 227 249
 | 250 253 288 289 291 295
-0.20 | claude-opus-4-20250514 | 11 16 32 39 47 50 57 64 69 71 77 104 122 151 152 154 162 165
 | 175 180 183 188 199 204 205 206 215 217 224 233 241 257 260 265
 | 287 292
 0.02 | o4-mini, deepseekr1-0528 | 6 9 23 24 25 31 35 46 53 59 62 67 68 79 95 96 100 103 109 113
 | 116 120 126 132 139 141 145 153 159 167 169 172 176 177 179 194
 | 210 213 216 239 240 245 246 252 255 256 264 268 269 271 279 285
 | 294 297 298
 0.25 | deepseek-reasoner | 5 17 28 43 66 75 90 99 124 125 128 131 201 208 231 234 247 251
 | 261 263 278

```



**Figure A52.** The task-information curve of proof masking over the entire observed ability range.

Model	Ability $\theta$	Model	Ability $\theta$
gemi-3.0-pro-preview	1.577	gemi-3.1-pro-preview	1.491
claude-opus-4-5-20251101	1.167	gpt-5.1	0.902
gpt-5	0.814	claude-sonnet-4.5-ext-thinking	0.803
grok-4	0.747	gpt-5.2	0.609
o3	0.606	deepseek-r1-0528	0.489
gemi-2.5-pro	0.404	deepseek-reasoner	0.358
o4-mini	0.301	o3-mini	-0.214
hermes-4	-0.238	deepseek-v3.1	-0.330
llama-3.1-405b	-0.356	claude-sonnet-4-20250514	-0.697
mistral-magistral	-0.783	claude-opus-4-20250514	-0.817
gpt-4o	-1.516	gpt-4.1	-1.569
grok-3	-1.819	deepseek-v3	-1.928

Table A35. Model  $\theta$  parameters estimated from PL3-PC results.

0.47		o3		0 2 4 7 12 21 33 42 49 72 85 87 91 97 105 115 130 155 171 189
				195 196 198 200 202 211 212 214 218 226 235 237 259 272 275 276
				277 280 283
0.69		gemi-2.5-pro, grok4, gpt-5, claude-sonnet-4-5-ext-thinking		10 13 74 80 102 110 111 147 178 181 184 220 242 262 273 274 282
				293 296
0.92		gpt-5.1		3 36 61 83 106 138 143 146 236 266 290
1.14		claude-opus-4-5-20251101, gpt-5.2		14 26 76 191 222 238 270
1.36		gemi-3-pro-preview		15 84 186 187 229 299
1.58				
1.81				173 284
2.03		gemi-3.1-pro-preview		

A line like this one:

-0.87		deepseek3, grok-3		48 51 56 58 63 78 98 114 137 168 203 230 244
-------	--	-------------------	--	----------------------------------------------

means that items 48, 51, . . . , 244, have difficulties  $b_j$ ; roughly around  $-0.87$ , or more precisely, that their estimated difficulties  $b_j$  fall in the bin centered around  $-0.87$ . This means models of ability  $\theta \approx -0.87$  (or more precisely, models whose estimated abilities fall into that same bin), such as DeepSeek-V3 and Grok-3, are just as likely to solve these problems as they are to fail them.

The shape of the Wright map is roughly the discrete counterpart of the task-information curve, rotated clockwise by 90 degrees. Where the items bunch up on the Wright map, the information curve tends to be high. The peak near 0 means that many items sit there, so the task becomes especially sensitive in that region. This obviously resembles a normal distribution centered at 0, but this is not because task information curves are inherently normal-shaped, nor is it because  $\theta$  is standardized to have mean 0 and variance 1. Several tasks in PROOFGRID have task-information curves that are skewed or irregular; see the task-information curve of PL4-PW as an example, Fig. A53. The near-normal appearance observed in Fig. A52 tells us something specific about the PL1-PM task items: their difficulties are concentrated around the middle of the ability scale, with increasingly fewer items as we move towards either tail end. Again, this matches the Wright map: item density is heaviest around the central region and thins out toward both ends. This means that the items of this task have difficulties that are nicely distributed on the  $\theta$  scale, and that the task's ability to distinguish models is also nicely distributed across that same central region, i.e., the task is most informative near the middle and its informativeness tapers off toward both ends. So this is a particularly well-balanced task, with items that are spread broadly across the ability range, instead of being piled up in one narrow hard or easy region. In simpler terms, this task has problems for weaker, middle, and stronger models, not just for the very strong or weak models.

This is one of the few PROOFGRID tasks that has good discriminating power in the lower half of the ability scale, with  $S_T(-2, 0.2) = 0.63$ .

## 6. PL3-PC IRT model

Fitting a 2PL IRT model on **PL3-PC** results yields the latent ability estimates shown in Table A35. We see that these estimates are quite consistent with the raw accuracies reported in Table 18, with a couple of minor exceptions.<sup>109</sup>

The following is the Wright map for this task:

PL3-PC MODEL-ITEM MAP: models and items on the same $\theta$ scale	
$\theta$ range: [-2.21, 2.49]	
$\theta$   Models	Items (b in bin)
-2.09	215
-1.85   deepseek3, grok-3	
-1.62   gpt-4o, gpt-4.1	
-1.38	
-1.15	
-0.92   claude-opus-4-20250514	7 8
-0.68   mistral-magistral, claude-sonnet-4-20250514	4 179
-0.45   deepseek3.1, llama_400b_nous 174	0 2 3 9 18 98 126 136 140 142 143 149 156 157 158 159 163 169 182 188 189 191 197 199 200 203 207 210 217 219 223 224 225 227 230 231 232 236 237 239 241 269 281
-0.21   o3-mini, hermes4 114	1 6 12 13 14 16 31 32 46 50 52 54 55 59 61 63 64 65 66 68 69 71 73 74 75 76 78 85 90 92 94 97 101 102 103 105 108 109 110 113 117 118 120 125 133 135 137 139 153 154 160 161 166 167 177 180 181 186 190 198 201 202 204 206 208 209 212 216 220 221 222 226 228 233 235 240 242 243 246 247 248 249 253 259 263 265 268 272 273 275 276 295 296 297 298
0.02   45	5 10 11 15 19 21 24 25 26 28 29 30 33 34 35 37 38 40 42 43 44 47 49 56 60 62 67 72 77 80 83 84 87 91 93 95 96 100 104 107 111 115 121 123 124 128 129 131 132 134 138 141 145 146 152 155 162 164 165 168 170 171 172 173 175 176 178 183 184 194 196 205 218 229 234 244 250 257 260 261 262 266 270 271 274 277 278 285 299
0.26   deepseek-reasoner, o4-mini 214	27 36 39 41 57 58 81 89 99 112 119 122 147 148 150 151 187 193 245 251 252 254 256 258 264 267 292
0.49   deepseekr1-0528, o3, gemini-2.5-pro, gpt-5.2	48 51 82 86 144 185 211 213 238 255 293 294
0.73   claude-sonnet-4-5-ext_thinking, gpt-5, grok4	23 53 70 192 279
0.96   gpt-5.1	17 20 22 79 106 127 284
1.20   claude-opus-4-5-20251101	116 130
1.43   gemini-3.1-pro-preview	88
1.66   gemini-3-pro-preview	
1.90	
2.13	290
2.37	282

We see that the task has items at many nearby difficulty levels across most of the respondent range, with a particularly dense item mass from about  $\theta \approx -0.45$  up to  $\theta \approx 0.49$ , plus additional coverage through 0.73 and 0.96. The map is clearly strongest in the lower-mid to upper-mid region, and thinner in the extreme tails. On the low end, there are only a couple of very easy items below  $\approx -0.9$ , and on the high end there are only a handful of hard items above  $\approx 1.2$ , with two ultra-hard items at 2.13 and 2.37 (items 290 and 282, the only 2 De Bruijn problems that are not failed by all models). So the task covers the middle region beautifully, but the very weakest and very strongest models are measured with fewer anchor items. This means the task should separate the broad population very well, as well as the mid-ability region, while the extreme-tail ordering is less stable.

The task is also good at separating weaker models (below 0 ability), as long as they are not too weak, e.g., we have  $S_T(-0.5, 0) = 0.81$ . For *very* weak models the discriminating power drops drastically, e.g.,  $S_T(-1.9, -0.6) = 0.21$ . Finally, as the Wright map suggests, **PL3-PC** is exceptionally good at separating mid-tier models, e.g., for  $\Theta = (-0.7, 0.7)$  we have  $S_T(\Theta) = 0.93$ .

<sup>109</sup> Gemini 3 here ranks ahead of Gemini 3.1, and GPT-5 ranks ahead of Claude Sonnet 4.5 with extended thinking, almost certainly due to the difficulty profiles (rather than the sheer number) of the problems solved by these models.

Next, we analyze some tasks that are better at discriminating top-tier models.

### 7. Type-indexed PL3-PW IRT models

We illustrate here with 2PL IRT models for two problem families that are useful for sharply discriminating the proof writing abilities of top-tier models: Tseitin-formula problems and relativized-pigeonhole problems. The following is the Wright map for PL3-PW Tseitin-formula problems:

PL3-PW TSEITIN-FORMULA MODEL-ITEM MAP: models and items on the same  $\theta$  scale  
 $\theta$  range: [-0.72, 3.12]

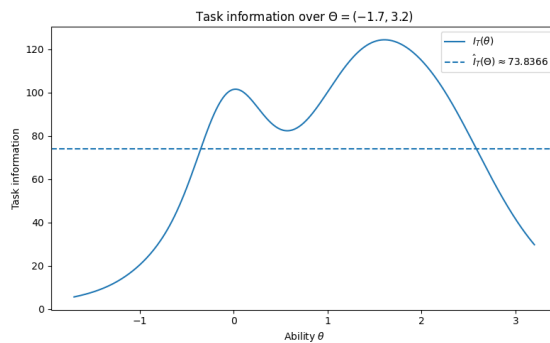
$\theta$	Models	Items (b in bin)
-0.63	mistral-magistral, deepseek3, gpt-4o, o4-mini, hermes4, deepseekr1-0528, gpt-4.1, o3, deepseek3.1, grok-3, claude-opus-4-20250514, o3-mini, llama_400b_nous	
-0.43		
-0.24	gemini-2.5-pro, claude-sonnet-4-5-ext_thinking, claude-sonnet-4-20250514	
-0.05		
0.14		
0.34		
0.53	gpt-5	23
0.72	grok4	45
0.91	claude-opus-4-5-20251101, deepseek-reasoner	
1.11	gpt-5.2	36
1.30	gpt-5.1	
1.49	gemini-3-pro-preview	44
1.68		39
1.87		8
2.07		32
2.26		35 38 47
2.45		
2.64		0 7 10 11 12 13 14 15 18 20 21 41 42 43 46 49
2.84		
3.03	gemini-3.1-pro-preview	1 4 26 27 31 34 37 40 59

This subtask is very good at discriminating top-tier models, with  $\mathcal{S}_T(0.9, 3.1) = 0.845$ , while it is completely ineffective at lower- and mid-tier discrimination. The picture for relativized pigeonhole problems is similar, with  $\mathcal{S}_T(0.9, 3.0) = 0.823$ :

PL3-PW RELATIVIZED PIGEONHOLE MODEL-ITEM MAP: models and items on the same  $\theta$  scale  
 $\theta$  range: [-0.60, 3.36]

$\theta$	Models	Items (b in bin)
-0.50	mistral-magistral, deepseek3, gpt-4o, o4-mini, hermes4, deepseekr1-0528, gpt-4.1, o3, claude-sonnet-4-5-ext_thinking, deepseek3.1, grok-3, claude-opus-4-20250514, o3-mini, llama_400b_nous, claude-sonnet-4-20250514	
-0.30		
-0.10	gemini-2.5-pro	
0.10	gpt-5, deepseek-reasoner	3
0.29		
0.49	gpt-5.1	
0.69	gemini-3-pro-preview	
0.89		14
1.08	claude-opus-4-5-20251101	7
1.28		10
1.48		4 8 9 18 22 24
1.68	grok4, gpt-5.2	6 12 17 27
1.88		11 16 19 20 23 25
2.07		
2.27		
2.47		5 13 15 21 26 28 29
2.67		
2.87		

Model	Ability $\theta$	Model	Ability $\theta$
gemini-3.1-pro-preview	3.268	gpt-5.2	1.577
gemini-3.0-pro-preview	0.915	grok-4	0.502
deepseek-reasoner	0.425	claude-opus-4-5-20251101	0.424
gpt-5	0.372	gpt-5.1	0.358
gemini-2.5-pro	0.307	o3	0.281
claude-sonnet-4.5-ext-thinking	0.206	deepseek-r1-0528	0.141
o4-mini	-0.151	o3-mini	-0.192
hermes-4	-0.388	deepseek-v3.1	-0.501
claude-opus-4-20250514	-0.508	claude-sonnet-4-20250514	-0.508
llama-3.1-405b	-0.730	deepseek-v3	-0.736
gpt-4o	-0.864	mistral-magistral	-1.213
gpt-4.1	-1.213	grok-3	-1.772



**Figure A53.** Ability parameters estimated from PL4-PW results, alongside the task-information curve of Hilbert proof-writing over the observed ability range. Information is low for models of ability  $\theta < 0$ .

3.06		
3.26		gemini-3.1-pro-preview

### 8. PL4-PW IRT model

The  $\theta$  values estimated from PL4-PW data are shown in the table of Fig. A53, next to the task-information curve over the observed ability range.

The following is the Wright map for PL4-PW:

PL4-PW MODEL-ITEM MAP: models and items on the same  $\theta$  scale  
 $\theta$  range: [-1.77, 3.27]

$\theta$	Models	Items (b in bin)
-1.65	grok-3	45 108
-1.39		
-1.14	mistral-magistral, gpt-4.1	
-0.89	gpt-4o	41 172
-0.64	deepseek3, llama_400b_nous	14 166
-0.39	hermes4, deepseek3.1, claude-opus-4-20250514, claude-sonnet-4-20250514	10 19 61 62 79
-0.13	o4-mini, o3-mini	24 75 115 127
0.12	deepseekr1-0528, claude-sonnet-4-5-ext_thinking	6 7 18 56 105 134 187
0.37	gemini-2.5-pro, claude-opus-4-5-20251101, gpt-5.1, gpt-5, o3, deepseek-reasoner	0 5 17 20 136 185
0.62	grok4	2 9
0.87	gemini-3-pro-preview	42 123 128 189
1.13		15 38 43 60 66 67 97 124 131 157 161
1.38		1 3 4 8 28 29 31 34 39 63 71 72 88 93 95 98 114 120 122 133 138
1.63	gpt-5.2	144 153 171 178 180 183 188 198
1.88		30 54 74 83
2.13		11 12 16 21 25 26 27 33 36 37 44 46 48 51 53 55 57 58 64 65 68
2.39		69 73
2.64		77 78 82 87 89 91 94 96 99 103 106 109 111 112 113 116 121 129
2.89		132
3.14	gemini-3.1-pro-preview	137 139 140 146 147 148 149 150 151 152 156 162 164 168 179 182
		186
		192 193 196 199
3.39		22 110 197
3.64		163

This task too is only effective for discriminating elite models. For the ability band  $\Theta = (0.5, 3.2)$ , which corresponds roughly to the models at or above the 83rd percentile of the observed ability range, the task has a normalized discrimination score of 0.7:  $\mathcal{S}_T(0.5, 3.2) = 0.7$ .



## Appendix C. Technical and Operational Details

This appendix summarizes the main operational details of the reported experiments: model identities, API routes, prompt formats, decoding and reasoning settings, confidence intervals for our results, batching policies, data generation, and evaluation procedures. We do not include the prompts here, since they tend to be fairly long; they can be found on [PROOFGRID's Github page](#).

### 1. Data Generation for PL<sub>1</sub>

Data generation for PL<sub>2</sub>, PL<sub>3</sub>, and PL<sub>4</sub> is already discussed in the paper. For PL<sub>1</sub>, premises and goals for all problems were generated randomly as ASTs (abstract syntax trees) by choosing a logical operator as the root (or an atom for a leaf) and then recursively generating subtrees up to a maximum depth. To minimize degenerate problems, we imposed the following constraints on any premises  $p_1, \dots, p_n$  and goal  $p$ :

1. all  $n + 1$  formulas must be distinct;
2.  $\{p_1, \dots, p_n\}$  must logically imply  $p$ ;
3. every premise  $p_i$  must be necessary, meaning that if we remove  $p_i$ , the goal  $p$  is no longer entailed by the remaining premises;
4. the goal does not contain any atoms that do not occur in the premises; and
5. the premises by themselves are logically consistent (as is the negation of the goal).

We used a SAT solver to enforce 2, 3, and 5.

We limit duplicate entries by normalizing a problem's representation as follows: sort the list of all premises, conjoin the resulting elements, form the conditional between that conjunction and the conclusion, and then universally quantify over all atoms in that conditional. We then identify two problems iff these universally quantified normal forms are alpha-equivalent (identical up to renaming of bound variables). Thus, for instance, the argument with  $p_1 = (A \Rightarrow B)$ ,  $p_2 = (B \Rightarrow C)$ ,  $p_3 = \neg C$ , and goal  $\neg A$  is represented by the QBF (Quantified Boolean Formula):

$$\forall A, B, C . (((A \Rightarrow B) \wedge (B \Rightarrow C) \wedge \neg C) \Rightarrow \neg A)$$

and is thus considered identical to the problem with  $p_1 = (C \Rightarrow D)$ ,  $p_2 = \neg E$ ,  $p_3 = (D \Rightarrow E)$ , and goal  $\neg C$ .

Terminological note: To improve readability, this paper often uses self-explanatory Latex-friendly notation for formulas instead of the plain-ASCII notation actually used in the tasks, e.g., writing  $(A \wedge \neg B)$  instead of  $(A \& \sim B)$ ,  $(p_1 \Rightarrow p_2)$  instead of  $(p_1 \implies p_2)$ ,  $(p_1 \vee p_2)$  instead of  $(p_1 | p_2)$ , and so on.

The Athena theorem prover that generated proofs for the **PL1-PC** problems uses a combination of backward and forward heuristics, ordered according to the strategy in Section 4.1.14 of the textbook *Fundamental Proof Methods in Computer Science* (Arkoudas and Musser 2017). The theorem prover is written as an Athena *method*, which is automatically instrumented so that a successful run produces a low-level *certificate*: a fully detailed proof comprising all applications of primitive inference rules that were made during the run, as well as all conditional (sub)proofs and (sub)proofs by contradiction, properly scoped. This certificate is typically a very long proof (many thousands of lines), because it includes all inferences made even during paths of the search tree that were ultimately dead ends. These proofs are then simplified by an aggressive proof-optimization algorithm (Arkoudas 2005) that removes detours and other inefficiencies from natural-deduction proofs, radically simplifying their structure.

The deliberate corruption of a proportion of these proofs used interventions such as altering the logical form of an argument to an inference rule (e.g., applying conjunction elimination to a disjunction), or the number of arguments, removing a step from a chain of subproofs, and so on.

## 2. Result Archives

We release result archives for all 15 tasks currently in PROOFGRID, in the form of one Yaml file per task:

	Task	Description
1	<b>PL1-PC</b>	Proof checking in PL <sub>1</sub>
2	<b>PL1-PC-c</b>	Proof checking in PL <sub>1</sub> , challenge version
3	<b>PL1-PW</b>	Proof writing in PL <sub>1</sub>
4	<b>PL1-PM</b>	Proof masking in PL <sub>1</sub>
5	<b>PL1-PM-c</b>	Proof masking in PL <sub>1</sub> , challenge version
6	<b>PL1-GF</b>	Proof gap-filling in PL <sub>1</sub>
7	<b>PL1-GF-c</b>	Proof gap-filling in PL <sub>1</sub> , challenge version
8	<b>PL2-PW</b>	Proof writing in PL <sub>2</sub>
9	<b>PL3-PC</b>	Proof checking in PL <sub>3</sub>
10	<b>PL3-PW</b>	Proof writing in PL <sub>3</sub>
11	<b>PL4-PW</b>	Proof writing in PL <sub>4</sub>
12	<b>PL4-PW-c</b>	Proof writing in PL <sub>4</sub> , challenge version
13	<b>EQ-PC</b>	Proof checking in EQ <sub>1</sub>
14	<b>EQ-ER</b>	Equation recovery in EQ <sub>1</sub>
15	<b>EQ-GF</b>	Gap filling in EQ <sub>1</sub>

The released Yaml files contain data documenting our results for all these tasks: problem descriptions and metadata, model identifiers, timestamps, and, for most runs, raw model outputs, processed model outputs, as well as partial token-usage and/or intermediate reasoning information.

Conceptually, a Yaml result file for task  $T$  is a list of result dictionaries, one per problem/item in  $T$ . Each such dictionary has a numeric index that serves as an ID, uniquely identifying that particular problem and associated result data in the context of  $T$ . For example, if our IRT analysis talks about “item 28” in **PL2-PW**, it means precisely that record  $r$  in the Yaml file for **PL2-PW** that has  $r['index'] == 28$ .

In addition, each result record typically has keys like `problem`, describing the input problem, metadata about the problem (e.g., whether or not the input proof to be checked is in fact correct), and a results dictionary mapping each of the 24 model identifiers  $M$  to a corresponding result dictionary for  $M$ . That model-specific dictionary contains a host of information about the response of  $M$  to the given problem: API call timestamps, the raw output that  $M$  generated, the processed/evaluated output (as carried out by our pipeline), evaluation results and details, and often a metadata key whose value indicates whether  $M$ 's response was ultimately counted as correct. As an example, here is (a truncated version of) item 0 in the results file for **PL1-PM**:

```

1 - index: 0
2 problem: '(C | A) # ((A ==> B) | (B & (B | A))) # (~C | B) # B'
3 conditionalized: true
4 proof: |-
5 assume h1 := ((C | A) & (((A ==> B) | (B & (B | A))) & (~C | B))) {
6 (C | A) BY left-and on h1;
7 ...
8 }
9 correctProof: true
10 complexProof: false
11 maskedProof: |-
12 assume h1 := ((C | A) & (((A ==> B) | (B & (B | A))) & (~C | B))) {
13 MASK38 BY MASK23 on MASK13;
14 ...
15 }
16 masks:
17 MASK1: (A ==> B)
18 MASK2: C
19 MASK3: (B & (B | A))
20 ...
21 llmMaskFillingResponses:
22 claude-opus-4-20250514:
23 sentTimestamp: '2025-08-20 14:05:51.817886+00:00'
24 receivedTimestamp: '2025-08-20 14:07:08.822000+00:00'

```

```

25 answer :
26 MASK1: (A ==> B)
27 MASK2: C
28 ...
29 unmaskedProof: |-
30 assume h1 := ((C | A) & (((A ==> B) | (B & (B | A))) & (~C | B))) {
31 (C | A) BY left-and on h1;
32 ...
33 correctMasks: false
34 instrumentedCheckerResult:
35 result: incorrect
36 errorType: notInAB
37 step: 7
38 offendingRule: claim
39 args:
40 - B
41 evaluatedArgs:
42 - B
43 missingFormula: B
44 fixes:
45 syntaxCorrections: []
46 structuralCorrections:
47 - - step-6
48 - " Overlooking a wrong conclusion that's equivalent up to conj/disj commutativity "
49 athenaAcceptedNonInstrumentedProof: false
50 athenaResponseToNonInstrumentedProof: |-
51 input:9:13: Error: Failed conclusion annotation.
52 The expected conclusion was:
53 (and C A)
54 but the obtained result was:
55 (and (or C A)
56 C).
57 evaluationTimestamp: '2026-01-07 16:05:49.353478'

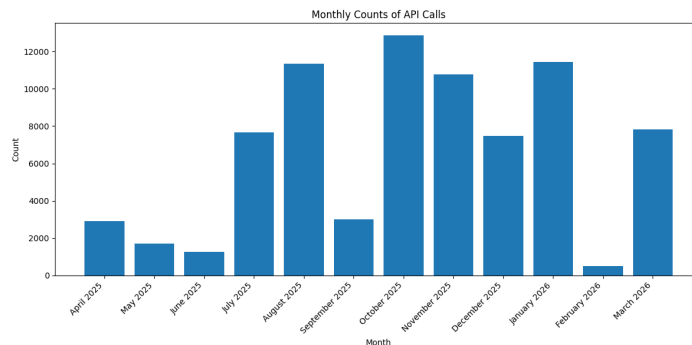
```

The problem field indicates the argument/sequent targeted by the relevant proof (the proof that was masked). The conditionalized flag indicates whether the problem was conditionalized (whether its premises were imported into the conclusion). The correctProof and complexProof flags are metadata about the starting proof: whether or not it solves the problem correctly and whether or not it has any complex/nested syntax forms. The value of maskedProof is the main problem-specific input given to the models, along with the general prompt for the task; it is obtained by masking the starting proof. The full mask assignment that recovers proof from maskedProof is given by the value of the masks dictionary. Model results are provided under llmMaskFillingResponses, a dictionary mapping our model identifiers to the full corresponding results. One such individual result dictionary, for Claude Opus 4, appears between lines 23 and 57 in the above listing (with truncated values for some of the keys). The dictionary contains timestamps (lines 23/24), the model's response (lines 25–28), the unmasked proof obtained by carrying out the assignments specified in the model's response (lines 29–32); a final correctMasks field indicating whether the model's response was ultimately correct; and detailed evaluation results in the dictionary instrumentedCheckerResult. The main payload of the latter is the result key, which can take the 3 values correct, incorrect, and (very rarely) unknown. If the result is incorrect, error details provided by the instrumented proof checker are given, including all syntax and/or semantic repairs.

Result files for other tasks are similarly structured.

### 3. Result Timeline and Interaction Protocol

The results were generated from April 2025 through March 2026; the distribution of API calls is shown in Fig. A54. All systems were queried as plain language models; we did not use external tools, retrieval, web search, or agent tooling. The standard interaction pattern was the single-call paradigm used by most classic benchmarks, without additional dialog turns. The reported outputs were not obtained by techniques like self-consistency, majority voting, or pass@k aggregation.



**Figure A54.** API call timeline.

#### 4. Model Identity and API Routing

We used several provider APIs. We routed each model request through the model’s native API when possible, relying on unified/shared API gateways for the rest (mostly OpenRouter, and occasionally Inworld).

OpenAI models from the GPT-5 family and from the o-series were not run through the same path as GPT-4o and GPT-4.1. The first used the [Responses API](#), while the second used [Chat Completions](#). Additionally, while our codebase contains OpenAI batch support and a synchronous parallel Gemini wrapper, our reported experiments used batch execution only for Anthropic models.

#### 5. Inputs and Outputs

Each task used its own prompt, typically in few-shot style with several ICL examples in the prompt. The prompts for all tasks can be found on the Github page. Most tasks require JSON outputs, the sole exception being gap filling for PL<sub>1</sub> (**PL1-GF**). There, a model generates a plain textual answer organized into labeled GAP-*i*: segments. Our evaluation pipeline then reconstructs a complete proof from that answer and checks it automatically. Table [A38](#) summarizes this information.

Family	Representative models	Primary API path	Batch mode	Default timeout
OpenAI reasoning models	gpt-5, gpt-5.1, gpt-5.2, o3, o3-mini, o4-mini	OpenAI Responses API in serial mode; OpenAI batch API available in the codebase	Serial in the reported runs	30 minutes
OpenAI chat models	gpt-4o, gpt-4.1	OpenAI Chat Completions API	Serial in the reported runs	30 minutes
Anthropic Claude models	claude-sonnet-4-20250514, claude-sonnet-4.5, claude-opus-4-20250514, claude-opus-4.5	Anthropic Messages API in serial mode; Anthropic Message Batches in batch mode	Batch used in the reported runs, with serial fill-ins	30 minutes
Google Gemini models	gemini-2.5-pro, gemini-3-pro-preview, gemini-3.1-pro-preview	Google genai client, generate_content	Serial in the reported runs	30 minutes
DeepSeek models	deepseek-v3, deepseek-v3.1, deepseek-r1, deepseek-reasoner	DeepSeek chat-completions-compatible API	No separate batch wrapper in the main benchmark path	30 minutes
xAI models	grok-3, grok-4, grok-4.20	xAI chat-completions-compatible API	No separate batch wrapper in the main benchmark path	30 minutes
OpenRouter-backed models	llama_400b_nous, hermes4, mistral-magistral, and related fallback models	OpenRouter chat-completions-compatible API	No provider batch path used here	30-second connect timeout; 30-minute read timeout

**Table A36.** Provider routing used. Model identifiers follow the naming scheme used in this paper; our public release artifacts expose the underlying provider identifiers as well.

ID used in this article	Queried model id	Run window
<i>OpenAI</i>		
gpt-5	gpt-5	Aug. 2025–Mar. 2026
gpt-5.1	gpt-5.1	Nov. 2025–Mar. 2026
gpt-5.2	gpt-5.2	Dec. 2025–Mar. 2026
o3-mini	o3-mini	Apr. 2025–Mar. 2026
o4-mini	o4-mini	Apr. 2025–Mar. 2026
o3	o3	Apr. 2025–Mar. 2026
gpt-4o	gpt-4o	Apr. 2025–Mar. 2026
gpt-4.1	gpt-4.1	Jun. 2025–Mar. 2026
<i>Google</i>		
gemini-3.1-pro-preview	gemini-3.1-pro-preview	Feb. 2026–Mar. 2026
gemini-2.5-pro	gemini-2.5-pro	Jun. 2025–Mar. 2026
gemini-3.0-pro-preview	gemini-3-pro-preview	Nov. 2025–Mar. 2026
<i>Anthropic</i>		
claude-sonnet-4-20250514	claude-sonnet-4-20250514	Jul. 2025–Mar. 2026
claude-sonnet-4.5-ext-thinking	claude-sonnet-4-5-20250929	Oct. 2025–Mar. 2026
claude-opus-4-20250514	claude-opus-4-20250514	Jul. 2025–Mar. 2026
claude-opus-4-5-20251101	claude-opus-4-5-20251101	Dec. 2025–Mar. 2026
<i>DeepSeek</i>		
deepseek-r1-0528	deepseekr1-0528	May 2025–Mar. 2026
deepseek-v3.1	deepseek/deepseek-chat-v3.1	Nov. 2025–Mar. 2026
deepseek-v3	deepseek/deepseek-chat-v3	Apr. 2025–Mar. 2026
deepseek-reasoner	deepseek-reasoner	Dec. 2025–Mar. 2026
<i>xAI</i>		
grok-4	grok-4-0709	Jul. 2025–Mar. 2026
grok-3	grok-3-latest	Jul. 2025–Mar. 2026
<i>Other OpenRouter-backed Models</i>		
mistral-magistral	mistralai/magistral-medium-2506	Jul. 2025–Oct. 2025
llama-3.1-405b	nousresearch/hermes-3-llama-3.1-405b	Apr. 2025–Mar. 2026
hermes-4	nousresearch/hermes-4-405b	Nov. 2025–Mar. 2026
<i>Inworld-routed models</i>		
mistral-magistral	mistralai/magistral-medium-latest	Feb. 2026–Mar. 2026

**Table A37.** The 24-model set reported in the paper, shown under the paper naming scheme together with the exact queried model identifiers and the observed run windows in the released archives.

## 6. Wilson Confidence Intervals

The parenthetical numbers in our tables are 95% confidence intervals for each reported accuracy, computed using the Wilson score interval for a binomial proportion. Concretely, for each model we treat every task item as a binary outcome: either the model got it right or it did not. Thus, if a model answers  $k$  of  $n$  items correctly, its observed accuracy is  $\hat{p} = k/n$ . Our tables report this point estimate together with the uncertainty due to the finite sample size. In the compact notation used in the tables, an entry such as 0.87 ( $\pm 0.02$ ) means that the estimated accuracy is 0.87 and the Wilson interval has

Task	Prompt style	Required output format
NDL/NDL <sub>0</sub> proof writing (PL <sub>1</sub> , PL <sub>2</sub> , PL <sub>3</sub> )	Few-shot	JSON dictionary containing at least a proof and a description/explanation of the proof reasoning
Hilbert proof writing (PL <sub>4</sub> )	Few shot	JSON dictionary containing a Hilbert proof and a short description
Proof checking	Few-shot	JSON dictionary containing a Boolean verdict and structured error information
Mask filling	Few-shot	JSON dictionary specifying the model's mask assignments
Gap filling	Few-shot	Free text containing one block per gap
Equational reasoning	Few shot	Structured JSON output (including confidence levels for EQ-GF)

**Table A38.** Prompt/input and output structure by task family.

Model group	Serial-mode policy	Batch-mode policy	Notes
gpt-5, gpt-5.1	Responses API; temperature not set; reasoning effort high; max_output_tokens=72K	Not used for the reported results	JSON-constrained tasks use the Responses-API text-format JSON path
gpt-5.2	Responses API; temperature not set; reasoning effort high; automatic extra-thinking budget of 64K on tasks that expose it; max_output_tokens=112K	Not used for the reported results	Used with the larger GPT-5 token budget path
OpenAI o-series (o3, o3-mini, o4-mini)	Responses API; temperature not set; reasoning effort high; max_output_tokens=72K	Not used for the reported results	The reported o-series results were effectively serial
claude-sonnet-4.5-ext-thinking	Anthropic Messages API; temperature omitted when thinking is enabled; thinking budget 12K; max_tokens=21,333	Anthropic Message Batches; thinking budget 16K; max_tokens=64K	Explicit extended thinking; stop reason is recorded when available in serial mode
claude-opus-4.5	Anthropic Messages API; temperature omitted when thinking is enabled; thinking budget 12K; max_tokens=21,333	Anthropic Message Batches; max_tokens=64K; effective thinking budget 32K after buffer reservation	Explicit extended thinking in both serial and batch paths
claude-sonnet-420250514	Anthropic Messages API; task temperature; no explicit thinking; max_tokens=21,333	Anthropic Message Batches; no explicit thinking; max_tokens=64K	No extended thinking in the reported runs
claude-opus-4.1	Anthropic Messages API; task temperature; no explicit thinking; max_tokens=21,333	Anthropic Message Batches; no explicit thinking; max_tokens=32K	No extended thinking in the reported runs
Gemini 2.5 / 3 / 3.1	Google genai client; provider defaults for temperature, output tokens, and reasoning (high "dynamic thinking level" by default)	Same provider defaults; "batch" runs are synchronous parallel direct calls	The benchmark task temperature is not passed into the Gemini wrapper
DeepSeek chat / reasoner	DeepSeek chat uses temperature 0.6 and top-p = 0.9; DeepSeek reasoner uses the provider reasoner path and enables JSON mode when requested; no explicit token cap	No separate batch wrapper in the main benchmark path	Task-level temperature is not propagated; these models use wrapper defaults
xAI Grok	Temperature 0.2; JSON response-format request when needed; no explicit reasoning-effort or output-token cap	No separate batch wrapper in the main benchmark path	Applies to grok-3, grok-4, and grok-4.20
OpenRouter-backed models	In the default raw-HTTP path: task temperature, max_tokens=32K, reasoning enabled with high effort; in the client-library path: task temperature with no explicit reasoning or token cap	No provider batch path used here	The gap-filling benchmark uses the client-library path; most other OpenRouter-backed tasks use the raw-HTTP path

**Table A39.** Compact summary of the main decoding, reasoning, and token-budget policies.

half-width 0.02, so the underlying 95% confidence interval is approximately [0.85, 0.89] after rounding to two decimals.

We used Wilson intervals rather than the simpler normal approximation because it behaves better for proportions near 0 or 1 and for moderate sample sizes. Our code uses the standard 95% Wilson formula with  $z = 1.96$ : it computes an adjusted center  $\tilde{p}$  and a half-width  $h$ , then forms the interval  $[\tilde{p} - h, \tilde{p} + h]$ . The numbers shown in parentheses are this half-width  $h$ , rounded to two decimal places; the code also computes the full lower and upper bounds explicitly. These intervals should be read in the usual frequentist way: if we repeated the same evaluation procedure many times on comparable samples, about 95% of the intervals constructed this way would contain the model's true accuracy on that task distribution.

## 7. Decoding, Reasoning, and Timeout Settings

Table A39 summarizes our main decoding and reasoning settings. Note that temperatures were not uniformly enforced, especially for the more advanced reasoning models (such as GPT-5x), which have deemphasized the use of temperature (and sampling parameters in general). Unless a setting is explicitly listed in the table, its value was left at the provider default.

### OpenAI models

Models from the GPT-5 family and the o-series were queried through the Responses API in serial mode. Temperature were not set explicitly for these models. Instead, we specified the values of reasoning parameters. GPT-5, GPT-5.1, GPT-5.2, GPT-5.4, and the o-series models used high-effort

settings. GPT-5.2 was typically given an automatic extra-thinking budget of 64K tokens in tasks that expose this field.

For gpt-4o and gpt-4.1, we used the Chat Completions API. These models therefore inherited our task-level temperature settings (reasoning parameters were inapplicable to these models), which are 0.6 for generative tasks like proof writing and 0.01 for discriminative tasks like proof checking.<sup>110</sup>

### **Anthropic Claude models.**

Serial Claude calls used the Anthropic Messages API. When explicit thinking was enabled, temperature values were omitted and thinking was requested directly. For the Claude 4.5 family in serial mode, we used a 12K thinking budget with a 21,333-token output cap. (These caps were imposed by Anthropic, which only supports higher values for these parameters in streaming mode.) Older Claude models were run without explicit thinking controls. Serial Claude calls also recorded the provider stop reason when available.

Batch Claude calls used Anthropic Message Batches. We requested provider-side thinking for the Claude 4.5 family and for explicit “extended thinking” aliases. In practice, this meant substantially larger batch-time output budgets than in serial mode. For example, the Opus 4.5 batch path used a provider-side 64K token cap together with a 32K internal thinking budget after reserving buffer space for the visible answer, whereas the Sonnet 4.5 batch path used a smaller fixed thinking budget.

### **Google Gemini models.**

Our Gemini calls used Google’s genai client. We did not explicitly set temperature, output-token limits, or reasoning controls, instead using provider defaults (high by default).

### **DeepSeek models.**

DeepSeek reasoner models were run through the provider’s reasoning-enabled path. When JSON output was requested, the wrapper also requested JSON formatting and enabled the provider’s explicit thinking mode. In both cases, the wrapper’s own defaults governed these calls rather than our task-level temperatures.

### **xAI models.**

xAI models were called through an API compatible with Chat Completions. We specified temperature values and when JSON output was needed, requested JSON formatting. We did not set reasoning-effort parameters or output-token caps.

### **OpenRouter-backed models.**

For models routed through OpenRouter, we used two related but distinct paths. In the default raw-HTTP path, we used the task-level temperature and requested JSON output whenever the task required it, capped output at 32K tokens, and enabled OpenRouter’s reasoning mode with high effort. The wrapper also used a 30-second connect timeout, a 30-minute read timeout, and one additional transport-level retry in the case of whitespace-only or apparently truncated HTTP bodies. In the client-library path, used only for **PL1-GF**, we did not specify reasoning or token-cap settings.

## **8. Batching, Partial Results, and Failure Policy**

In the reported experiments, the only models run in batch mode were Anthropic models. These were run mostly through Anthropic Message Batches, with serial fill-ins used in most tasks to complete missing items or rerun failures. Batched outputs were matched back to problems through explicit custom identifiers rather than through positional order.

<sup>110</sup> While 0.6 might seem high for a reasoning task, we have empirically found that it outperforms lower temperatures for proof writing. In general, proof checking is a low-entropy judgment task, while proof writing is a high-entropy search task. Low temperature helps the former by reducing noise, while moderate temperature helps the latter by improving search over possible proof trajectories.

Malformed outputs were generally not discarded silently. Instead, we preserved them either as raw outputs or through explicit flags such as `illFormattedProofOutput` or `illFormattedResponse`, or via task-specific parse-failure markers. Proof-checking tasks also recorded explicit API-call failures. These malformed or missing outputs remain visible in the released output files.

Our rerun policy was necessarily variable across models. Many models returned valid outputs consistently and required little manual intervention. Others, especially some reasoning-oriented models on difficult tasks, frequently exhausted their output budget, returned malformed answers, or resulted in other API call failures. Grok-4 was especially difficult in this respect, and some advanced GPT models also often declined to return an answer on the hardest problems. For such models, we used a best-effort fill-in policy: malformed outputs and API-level failures were rerun, typically at least three times and sometimes more.

If a problem instance did not yield any response (e.g., because of a server-side API-call error), that was not counted against the model (the data point was removed from the denominator), but if an incomplete or ill-formed response was returned, it was counted as an answer. When possible, invalid or partial results were either cleaned with postprocessing or the API calls were repeated.

## 9. Evaluation

Performance is not computed directly from raw model outputs. Each task has a deterministic evaluation pipeline, which forms an important part of the experimental design. These are described in our discussions of the individual tasks and released in full operational detail in the Github code base.

## 10. Reasoning Metadata in the Released Result Files

Reasoning metadata is present with limited coverage. Many GPT- and Claude-based runs preserve one or more of the following:

- provider usage summaries;
- extracted reasoning summaries;
- stored thinking traces;
- `thinkingTokens`;
- `outputTokens`;
- provider stop reasons.

However, these fields are not uniformly available across all providers and all tasks. They are common in several proof-checking runs and in subsets of the masking, gap-filling, and equational results, but they are less common for Gemini, xAI, and some OpenRouter-backed models.

## References

- Snell, C.V.; Lee, J.; Xu, K.; Kumar, A. Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning. In Proceedings of the The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025, 2025.
- Glazer, E.; Erdil, E.; Besiroglu, T.; Chicharro, D.; Chen, E.; Gunning, A.; Olsson, C.F.; Denain, J.S.; Ho, A.; de Oliveira Santos, E.; et al. FrontierMath: A Benchmark for Evaluating Advanced Mathematical Reasoning in AI, 2024, [[arXiv:cs.AI/2411.04872](https://arxiv.org/abs/2411.04872)].
- Sternberg, Robert., Ed. *The Cambridge Handbook of Intelligence*, 2nd ed.; Cambridge University Press, 2020.
- Liang, P.; Bommasani, R.; Lee, T.; Tsipras, D.; Soylu, D.; Yasunaga, M.; Zhang, Y.; Narayanan, D.; Wu, Y.; Kumar, A.; et al. Holistic evaluation of language models. <https://openreview.net/pdf?id=iO4LZibEqW>, 2023.
- Groarke, L. Deductivism Within Pragma-Dialectics. *Argumentation* **1999**, *13*.
- Musgrave, A. Popper and hypothetico-deductivism. In *Karl Popper: Critical Assessments of Leading Philosophers*; Miller, D.M., Ed.; Routledge, 2003; Vol. 2, pp. 205–234.
- Musgrave, A. Deductivism Surpassed: Or, Foxing in its Margins? *Journal for General Philosophy of Science* **2012**, *43*, 125–132.
- Heim, I.; Kratzer, A. *Semantics in generative grammar*; Oxford [u.a.]: Blackwell, 1998.
- Tversky, A.; Kahneman, D. Judgment under Uncertainty: Heuristics and Biases. *Science* **1974**, *185*, 1124–1131.
- MacCartney, B. Natural Language Inference. PhD thesis, Stanford, 2009.

- Dagan, I.; Dolan, B.; Magnini, B.; Roth, D. Recognizing textual entailment: Rational, evaluation and approaches. *Natural Language Engineering* **2009**, *15*, i–xvii.
- Bowman, S.R.; Angeli, G.; Potts, C.; Manning, C.D. A large annotated corpus for learning natural language inference. In Proceedings of the Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing; Màrquez, L.; Callison-Burch, C.; Su, J., Eds., Lisbon, Portugal, Sep 2015; pp. 632–642. <https://doi.org/10.18653/v1/D15-1075>.
- Williams, A.; Nangia, N.; Bowman, S. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In Proceedings of the Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). Association for Computational Linguistics, 2018, pp. 1112–1122.
- Manning, C.D. Local Textual Inference: It's hard to circumscribe, but you know it when you see it – and NLP needs it.
- Bos, J.; Markert, K. Recognising Textual Entailment with Logical Inference. In Proceedings of the Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing; Mooney, Raymond and Brew, Chris and Chien, Lee-Feng and Kirchoff, Katrin., Ed. Association for Computational Linguistics, Oct 2005, pp. 628–635.
- Akhmatova, E.; Mollá, D. Recognizing textual entailment via atomic propositions. In Proceedings of the Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment, Berlin, Heidelberg, 2005; p. 385–403.
- Bos, J.; Markert, K. When logical inference helps determining textual entailment (and when it doesn't).
- Kamp, H. A Theory of Truth and Semantic Representation. In *Formal Methods in the Study of Language*; Groenendijk, J.A.G.; Janssen, T.M.V.; Stokhof, M.B.J., Eds.; Mathematisch Centrum: Amsterdam, 1981; Vol. 1, pp. 277–322.
- Copestake, A.; Flickinger, D.; Pollard, C.; Sag, I. Minimal recursion semantics: An introduction. *Research on Language and Computation* **2005**, *3*, 281–332.
- Clark, P.; Tafjord, O.; Richardson, K. Transformers as soft reasoners over language. In Proceedings of the Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, 2021, IJCAI 2020.
- Tafjord, O.; Dalvi, B.; Clark, P. ProofWriter: Generating Implications, Proofs, and Abductive Statements over Natural Language. In Proceedings of the Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021; Zong, C.; Xia, F.; Li, W.; Navigli, R., Eds. Association for Computational Linguistics, Aug 2021, pp. 3621–3634.
- Saparov, A.; He, H. Language Models Are Greedy Reasoners: A Systematic Formal Analysis of Chain-of-Thought. In Proceedings of the International Conference on Learning Representations, 2023.
- Ontañón, S.; Ainslie, J.; Cvicek, V.; Fisher, Z. LogicInference: A New Dataset for Teaching Logical Inference to seq2seq Models. In Proceedings of the Proceedings of ICLR 2022 workshop on Objects, Structure and Causality, 2022.
- Wan, Y.; Wang, W.; Yang, Y.; Yuan, Y.; Huang, J.t.; He, P.; Jiao, W.; Lyu, M. LogicAsker: Evaluating and Improving the Logical Reasoning Ability of Large Language Models. In Proceedings of the Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing; Al-Onaizan, Y.; Bansal, M.; Chen, Y.N., Eds., Miami, Florida, USA, Nov 2024; pp. 2124–2155. <https://doi.org/10.18653/v1/2024.emnlp-main.128>.
- Tian, J.; Li, Y.; Chen, W.; Xiao, L.; He, H.; Jin, Y. Diagnosing the First-Order Logical Reasoning Ability Through LogicNLI. In Proceedings of the Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing; Moens, M.F.; Huang, X.; Specia, L.; Yih, S.W.t., Eds. Association for Computational Linguistics, Nov 2021, pp. 3738–3747.
- Morishita, T.; Morio, G.; Yamaguchi, A.; Sogawa, Y. Learning Deductive Reasoning from Synthetic Corpus based on Formal Logic. In Proceedings of the Proceedings of the 40th International Conference on Machine Learning (ICML); Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; Scarlett, J., Eds. PMLR, Jul 2023, Vol. 202, *Proceedings of Machine Learning Research*, pp. 25254–25274.
- Han, S.; Schoelkopf, H.; Zhao, Y.; Qi, Z.; Riddell, M.; Zhou, W.; Coady, J.; Peng, D.; Qiao, Y.; Benson, L.; et al. FOLIO: Natural Language Reasoning with First-Order Logic. In Proceedings of the Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing; Al-Onaizan, Y.; et al., Eds., Miami, Florida, USA, Nov 2024; pp. 22017–22031.
- Zellers, R.; Holtzman, A.; Bisk, Y.; Farhadi, A.; Choi, Y. HellaSwag: Can a Machine Really Finish Your Sentence? In Proceedings of the Proceedings of the 57th Annual Meeting of the Association for Computational

- Linguistics; Korhonen, A.; Traum, D.; Màrquez, L., Eds., Florence, Italy, Jul 2019; pp. 4791–4800. <https://doi.org/10.18653/v1/P19-1472>.
- Cobbe, K.; Kosaraju, V.; et al. Training Verifiers to Solve Math Word Problems. In Proceedings of the Advances in Neural Information Processing Systems, 2021.
- Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; Steinhardt, J. Measuring Mathematical Problem Solving With the MATH Dataset. *arXiv preprint arXiv:2103.03874* 2021.
- He, C.; Luo, R.; Bai, Y.; Hu, S.; Thai, Z.; Shen, J.; Hu, J.; Han, X.; Huang, Y.; Zhang, Y.; et al. OlympiadBench: A Challenging Benchmark for Promoting AGI with Olympiad-Level Bilingual Multimodal Scientific Problems. In Proceedings of the Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers); Ku, L.W.; Martins, A.; Srikumar, V., Eds. Association for Computational Linguistics, Aug 2024, pp. 3828–3850.
- Bofei, G.; et al. Omni-MATH: A Universal Olympiad Level Mathematic Benchmark For Large Language Models, 2024, [arXiv:cs.CL/2410.07985].
- Sun, H.; Min, Y.; Chen, Z.; Zhao, W.X.; Liu, Z.; Wang, Z.; Fang, L.; Wen, J.R. Challenging the Boundaries of Reasoning: An Olympiad-Level Math Benchmark for Large Language Models, 2025, [arXiv:cs.CL/2503.21380].
- Nguyen, M.V.; Luo, L.; Shiri, F.; Phung, D.; Li, Y.F.; Vu, T.T.; Haffari, G. Direct Evaluation of Chain-of-Thought in Multi-hop Reasoning with Knowledge Graphs. In Proceedings of the Findings of the Association for Computational Linguistics: ACL 2024; Ku, L.W.; Martins, A.; Srikumar, V., Eds., Bangkok, Thailand, Aug 2024; pp. 2862–2883. <https://doi.org/10.18653/v1/2024.findings-acl.168>.
- Li, J.; Beeching, E.; Tunstall, L.; Lipkin, B.; Soletskyi, R.; Huang, S.; Rasul, K.; Yu, L.; Jiang, A.Q.; Shen, Z.; et al. NuminaMath: The largest public dataset in AI4Maths with 860k pairs of competition math problems and solutions. [http://faculty.bicmr.pku.edu.cn/~dongbin/Publications/numina\\_dataset.pdf](http://faculty.bicmr.pku.edu.cn/~dongbin/Publications/numina_dataset.pdf), 2024.
- Clark, P.; Cowhey, I.; Etzioni, O.; Khot, T.; Sabharwal, A.; Schoenick, C.; Tafjord, O. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv:1803.05457v1* 2018.
- Zhang, H.; Li, L.H.; Meng, T.; Chang, K.W.; Van Den Broeck, G. On the paradox of learning to reason from data. In Proceedings of the Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, 2023, IJCAI '23. <https://doi.org/10.24963/ijcai.2023/375>.
- Mahdavi, H.; Hashemi, A.; Daliri, M.; Mohammadipour, P.; Farhadi, A.; Malek, S.; Yazdanifard, Y.; Khasahmadi, A.; Honavar, V. Brains vs. Bytes: Evaluating LLM Proficiency in Olympiad Mathematics, 2025, [arXiv:cs.AI/2504.01995].
- Suzgun, M.; et al. Challenging BIG-Bench Tasks and Chain-of-Thought. In Proceedings of the International Conference on Machine Learning, 2023.
- Google Research. BIG-Bench Extra Hard, 2025, [arXiv:cs.CL/2502.19187].
- Hendrycks, D.; Burns, C.; Basart, S.; Zou, A.; Mazeika, M.; Song, D.; Steinhardt, J. Measuring Massive multitask language understanding. In Proceedings of the International Conference on Learning Representations, 2021.
- Wang, Y.; Ma, X.; Zhang, G.; Ni, Y.; Chandra, A.; Guo, S.; Ren, W.; Arulraj, A.; He, X.; Jiang, Z.; et al. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark, 2024, [arXiv:cs.CL/2406.01574].
- Scale AI and Center for AI Safety. Humanity's Last Exam, 2025. A benchmark designed to test AI systems at the frontier of human knowledge.
- Team, M.A.P.; Du, X.; Yao, Y.; Ma, K.; Wang, B.; Zheng, T.; Zhu, K.; Liu, M.; Liang, Y.; Jin, X.; et al. SuperGPQA: Scaling LLM Evaluation across 285 Graduate Disciplines, 2025, [arXiv:cs.CL/2502.14739].
- Chollet, F. On the Measure of Intelligence. *arXiv preprint arXiv:1911.01547* 2019.
- Spelke, E.S.; Kinzler, K.D. Core knowledge. *Developmental Science* 2007, 10, 89–96.
- Spelke, E.S. Core knowledge. *American Psychologist* 2000, 55, 1233–1243. <https://doi.org/10.1037/0003-066X.55.11.1233>.
- Chollet, F.; Knoop, M.; Kamradt, G.; Landers, B.; Pinkard, H. ARC-AGI-2: A New Challenge for Frontier AI Reasoning Systems, 2026, [arXiv:cs.AI/2505.11831].
- Spearman, C. "General Intelligence," Objectively Determined and Measured. *The American Journal of Psychology* 1904, 15, 201–292.
- Cattell, R.B. The measurement of adult intelligence. *Psychological Bulletin* 1943, 40, 153–193. <https://doi.org/10.1037/h0059973>.
- Cattell, R.B. Theory of fluid and crystallized intelligence: A critical experiment. *Journal of Educational Psychology* 1963, 54, 1–22. <https://doi.org/10.1037/h0046743>.

- Carroll, J.B. *Human Cognitive Abilities: A Survey of Factor-Analytic Studies*; Cambridge University Press: Cambridge, England, 1993. <https://doi.org/10.1017/CBO9780511571312>.
- McCallum, R. *Handbook of Nonverbal Assessment*; Kluwer Academic/Plenum Publishers, 2003.
- Gignac, G.E. Raven's is not a pure measure of general intelligence: Implications for g factor theory and the brief measurement of g. *Intelligence* **2015**, *52*, 71–79. <https://doi.org/10.1016/j.intell.2015.07.006>.
- Walrath, R.; Willis, J.O.; Dumont, R.; Kaufman, A.S. Factor-Analytic Models of Intelligence. In *The Cambridge Handbook of Intelligence*, 2nd ed.; Sternberg, R.J., Ed.; Cambridge University Press: Cambridge, 2020; pp. 75–98. <https://doi.org/10.1017/9781108770422.006>.
- Wechsler, D. *The Measurement of Adult Intelligence*; Williams & Wilkins Co.: Baltimore, MD, 1939. <https://doi.org/10.1037/10020-000>.
- Lakin, J.M.; Kell, H.J. Intelligence and Reasoning. In *The Cambridge Handbook of Intelligence*, 2nd ed.; Sternberg, R.J., Ed.; Cambridge University Press: New York, NY, 2020; pp. 528–552. <https://doi.org/10.1017/9781108770422.023>.
- McGrew, K.S.; Schneider, W.J.; Decker, S.L.; Bulut, O. A Psychometric Network Analysis of CHC Intelligence Measures: Implications for Research, Theory, and Interpretation of Broad CHC Scores “Beyond g”. *Journal of Intelligence* **2023**, *11*, 19. <https://doi.org/10.3390/jintelligence11010019>.
- McGrew, K.S. CHC theory and the human cognitive abilities project: Standing on the shoulders of the giants of psychometric intelligence research. *Intelligence* **2009**, *37*, 1–10.
- Beauducel, A.; Kersting, M. Fluid and Crystallized Intelligence and the Berlin Model of Intelligence Structure (BIS). *European Journal of Psychological Assessment* **2002**, *18*, 97–112. <https://doi.org/10.1027//1015-5759.18.2.97>.
- Sloutsky, V.M. From Perceptual Categories to Concepts: What Develops? *Cognitive Science* **2010**, *34*, 1244–1286. <https://doi.org/10.1111/j.1551-6709.2010.01129.x>.
- Bracken, B.A.; McCallum, R.S. *Universal Nonverbal Intelligence Test*; Riverside Publishing: Itasca, IL, 1998.
- Wang, X.; Huang, J.; Zhang, X.; Wang, T.; Ma, J.W. Your Reasoning Benchmark May Not Test Reasoning: Revealing Perception Bottleneck in Abstract Reasoning Benchmarks, 2026, [\[arXiv:cs.CL/2512.21329\]](https://arxiv.org/abs/2512.21329).
- Chollet, F.; Knoop, M.; Kamradt, G.; Landers, B. ARC Prize 2024: Technical Report, 2025, [\[arXiv:cs.AI/2412.04604\]](https://arxiv.org/abs/2412.04604).
- Zheng, K.; Han, J.M.; Polu, S. MiniF2F: a cross-system benchmark for formal Olympiad-level mathematics, 2022, [\[arXiv:cs.AI/2109.00110\]](https://arxiv.org/abs/2109.00110).
- Baek, Shaun Lee et al.. Rosetta-PL: Propositional Logic as a Benchmark for Large Language Model Reasoning. In Proceedings of the Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 4: Student Research Workshop), Albuquerque, USA, Apr 2025; pp. 551–562.
- Weston, J.; Bordes, A.; Chopra, S.; Rush, A.M.; van Merriënboer, B.; Joulin, A.; Mikolov, T. Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks. In Proceedings of the International Conference on Learning Representations (ICLR), 2016.
- Rein, D.; Hou, B.L.; Stickland, A.C.; Petty, J.; Pang, R.Y.; Dirani, J.; Michael, J.; Bowman, S.R. GPQA: A Graduate-Level Google-Proof Q&A Benchmark, 2023, [\[arXiv:cs.AI/2311.12022\]](https://arxiv.org/abs/2311.12022).
- Dalvi, B.; Jansen, P.; Tafjord, O.; Xie, Z.; Smith, H.; Pipatanangkura, L.; Clark, P. Explaining Answers with Entailment Trees. In Proceedings of the Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing; Moens, M.F.; Huang, X.; Specia, L.; Yih, S.W.t., Eds., Online and Punta Cana, Dominican Republic, Nov 2021; pp. 7358–7370. <https://doi.org/10.18653/v1/2021.emnlp-main.585>.
- Chen, W.; Yin, M.; Ku, M.; Lu, P.; Wan, Y.; Ma, X.; Xu, J.; Wang, X.; Xia, T. TheoremQA: A Theorem-driven Question Answering Dataset. In Proceedings of the Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing; Bouamor, H.; Pino, J.; Bali, K., Eds., Singapore, Dec 2023; pp. 7889–7901.
- Johnson-Laird, P.N. Inference and Mental Models. In *Perspectives on Thinking and Reasoning*; Newstead, S.E.; Evans, J.S.T., Eds.; Lawrence Erlbaum: Hillsdale NJ, 1995; pp. 115–146.
- Ballon, M.; Algaba, A.; Ginis, V. The Relationship Between Reasoning and Performance in Large Language Models – o3 (mini) Thinks Harder, Not Longer, 2025, [\[arXiv:cs.LG/2502.15631\]](https://arxiv.org/abs/2502.15631).
- Liu, H.; Liu, J.; Cui, L.; Teng, Z.; Duan, N.; Zhou, M.; Zhang, Y. LogiQA 2.0—An Improved Dataset for Logical Reasoning in Natural Language Understanding. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* **2023**, *31*, 2947–2962.
- Richardson, K.; Sabharwal, A. Pushing the Limits of Rule Reasoning in Transformers through Natural Language Satisfiability. In Proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence. AAAI Press, 2022, pp. 11209–11219.

- Parmar, Mihir et al.. LogicBench: Towards Systematic Evaluation of Logical Reasoning Ability of Large Language Models. In Proceedings of the Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Bangkok, Thailand, Aug 2024; pp. 13679–13707.
- Patel, N.; Kulkarni, M.; Parmar, M.; Budhiraja, A.; Nakamura, M.; Varshney, N.; Baral, C. Multi-LogiEval: Towards Evaluating Multi-Step Logical Reasoning Ability of Large Language Models. In Proceedings of the Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing; Al-Onaizan, Y.; Bansal, M.; Chen, Y.N., Eds., Miami, Florida, USA, Nov 2024; pp. 20856–20879. <https://doi.org/10.18653/v1/2024.emnlp-main.1160>.
- Albalak, A.; Phung, D.; Lile, N.; Rafailov, R.; Gandhi, K.; Castricato, L.; Singh, A.; Blagden, C.; Xiang, V.; Mahan, D.; et al. Big-Math: A Large-Scale, High-Quality Math Dataset for Reinforcement Learning in Language Models, 2025, [arXiv:cs.LG/2502.17387].
- Kaliszyk, C.; Chollet, F.; Szegedy, C. HolStep: A Machine Learning Dataset for Higher-order Logic Theorem Proving. In Proceedings of the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings, 2017.
- Yang, K.; Swope, A.M.; Gu, A.; Chalamala, R.; Song, P.; Yu, S.; Godil, S.; Prenger, R.; Anandkumar, A. LeanDojo: theorem proving with retrieval-augmented language models. In Proceedings of the Proceedings of the 37th International Conference on Neural Information Processing Systems, Red Hook, NY, USA, 2023; NIPS '23.
- Welleck, S.; Liu, J.; Bras, R.L.; Hajishirzi, H.; Choi, Y.; Cho, K. NaturalProofs: Mathematical Theorem Proving in Natural Language. In Proceedings of the Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual; Vanschoren, J.; Yeung, S., Eds., 2021.
- Welleck, S.; Liu, J.; Lu, X.; Hajishirzi, H.; Choi, Y. NATURALPROVER: grounded mathematical proof generation with language models. In Proceedings of the Proceedings of the 36th International Conference on Neural Information Processing Systems, Red Hook, NY, USA, 2022; NIPS '22.
- Yousefzadeh, R.; Cao, X. A Lean Dataset for International Math Olympiad: Small Steps towards Writing Math Proofs for Hard Problems. *Transactions on Machine Learning Research* 2025.
- Arkoudas, K. Type- $\omega$  DPLs. AI Memo 2001-27, MIT AI Lab, 2001. Available from: <ftp://publications.ai.mit.edu/ai-publications/2001/AIM-2001-027.pdf>.
- Barnett, S.M.; Ceci, S.J. When and where do we apply what we learn?: A taxonomy for far transfer. *Psychological Bulletin* 2002, 128, 612–637. <https://doi.org/10.1037/0033-2909.128.4.612>.
- Wang, J.; Chen, Y. *Introduction to Transfer Learning: Algorithms and Practice*; Machine Learning: Methodologies and Applications, Springer, 2023.
- Schickore, J. Scientific Discovery. In *The Stanford Encyclopedia of Philosophy*, Fall 2025 ed.; Zalta, E.N.; Nodelman, U., Eds.; Metaphysics Research Lab, Stanford University, 2025.
- Baker, C.L.; Saxe, R.; Tenenbaum, J.B. Action understanding as inverse planning. *Cognition* 2009.
- Ramirez, M.; Geffner, H. Probabilistic Plan Recognition Using Off-The-Shelf Classical Planners. In Proceedings of the Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), 2010.
- Buss, S.; Nordström, J. Proof Complexity and SAT Solving. In *Handbook of Satisfiability*; IOS Press: Amsterdam, 2009; pp. 233–350.
- Arkoudas, K.; Musser, D. *Fundamental Proof Methods in Computer Science*; MIT Press: Cambridge, MA, 2017.
- Sinz, C. Towards an optimal CNF encoding of Boolean cardinality constraints. In Proceedings of the Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, Berlin, Heidelberg, 2005; p. 827–831. [https://doi.org/10.1007/11564751\\_73](https://doi.org/10.1007/11564751_73).
- Lauria, M.; Elffers, J.; Nordström, J.; Vinyals, M. CNFgen : A generator of crafted benchmarks. In Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing, SAT 2017, 2017, Vol. 10491, *Lecture Notes in Computer Science*, pp. 464–473. <https://github.com/MassimoLauria/cnfgem>, [https://doi.org/10.1007/978-3-319-66263-3\\_30](https://doi.org/10.1007/978-3-319-66263-3_30).
- Miksa, M.; Nordström, J. Long Proofs of (Seemingly) Simple Formulas. In Proceedings of the Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT 2014). Springer, 2014, Vol. 8561, *Lecture Notes in Computer Science*, pp. 121–137.
- Kovács, Laura and Voronkov, Andrei. First-Order Theorem Proving and Vampire. In Proceedings of the Computer Aided Verification, Berlin, Heidelberg, 2013; pp. 1–35.
- Sutcliffe, G.; Schulz, S.; Claessen, K.; Gelder, A.V. Using the TPTP Language for Writing Derivations and Finite Interpretations. In Proceedings of the Automated Reasoning, Third International Joint Conference, IJCAR

- 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings; Furbach, U.; Shankar, N., Eds. Springer, 2006, Vol. 4130, *Lecture Notes in Computer Science*, pp. 67–81.
- Arkoudas, K. Simplifying Proofs in Fitch-Style Natural Deduction Systems. *Journal of Automated Reasoning* **2005**, *34*, 239–294.
- Hindley, J.; Lercher, B.; Seldin, J. *Introduction to Combinatory Logic*; Lecture note series, 7, Cambridge University Press, 1972.
- Athena Language Team. Athena: A Language for Proof Engineering. <https://athena-lang.org>, 2025.
- Mendelson, E. *Introduction to Mathematical Logic*, 5th ed.; Discrete Mathematics and Its Applications, Chapman and Hall/CRC: New York, 2010.
- Tarski, A. Über einige fundamentale Begriffe der Metamathematik. *Comptes Rendus des Séances de la Société des Sciences et des Lettres de Varsovie, Classe III* **1930**, *23*, 22–29. English translation in Tarski (1956), *Logic, Semantics, Metamathematics*.
- Herbrand, J. Recherches sur la théorie de la démonstration. *Travaux de la Société des Sciences et des Lettres de Varsovie* **1930**, *III*, *33*, 33–160. (1971) *Logical Writings*. Harvard University Press and Reidel.
- Birkhoff, Garrett. On the structure of abstract algebras. In *Proceedings of the Mathematical proceedings of the Cambridge philosophical society*. Cambridge University Press, 1935, Vol. 31, pp. 433–454.
- Bengio, Y.; Courville, A.; Vincent, P. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1798–1828.
- Zheng, L.; Chiang, W.L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.; et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Proceedings of the Advances in Neural Information Processing Systems*; Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; Levine, S., Eds. Curran Associates, Inc., 2023, Vol. 36, pp. 46595–46623.
- Hambleton, R.K.; Swaminathan, H.; Rogers, H.J. *Fundamentals of Item Response Theory*; SAGE: Newbury Park, CA, 1991.
- Reise, S.P.; Revicki, D.A., Eds. *Handbook of Item Response Theory Modeling: Applications to Typical Performance Assessment*; Routledge: New York, NY, 2014.
- Hambleton, R.; Swaminathan, H. *Item Response Theory: Principles and Applications*; Humanities, Social Sciences and Law, Springer Netherlands, 2013.
- Ly, A.; Myung, J.; Pitt, M. A tutorial on Fisher information. *Journal of Mathematical Psychology* **2017**, *80*, 40–55. <https://doi.org/10.1016/j.jmp.2017.05.002>.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.