

Article

Not peer-reviewed version

A Declared World Model: Language Models Write the Rules of the Scene They Are Shown

[Jean-Jacques Dubray](#)*

Posted Date: 19 October 2026

Keywords: world models; large language models; neurosymbolic AI; open-world digital twins; partial observability; rule learning; joint embedding predictive architecture



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

A Declared World Model: Language Models Write the Rules of the Scene They Are Shown

Jean-Jacques Dubray

Cognitive Fab; jdubray@gmail.com

Abstract

LeCun's architecture for autonomous machine intelligence places a learned joint-embedding predictive world model at its centre. We keep its six modules with two departures: the world model is declared rather than learned, and a language model serves as configurator. The configurator writes the rules of the world from a labelled scene graph, and a deterministic engine runs them, accepting or refusing each proposed action; every recovered world is scored against an exact answer key. We ask whether the rules are written for the world or for the picture. From one scene, every configurator tested writes correct rules bound to the objects and numbers in view, and no correct world holds on a layout with a second door. Separating rules from facts, asking for rules over kinds of objects (which lifts only the strongest models), or showing the same world under several layouts lifts the rules over kinds of objects; variation lifts exactly the factors that vary, the doors when the doors vary and the wall when the wall varies, in line with the sufficient-variation condition for learned representations. Because the world model is declared, what each rule is bound to can be read from its text and checked before the world is used.

Keywords: world models; large language models; neurosymbolic AI; open-world digital twins; partial observability; rule learning; joint embedding predictive architecture

1. Introduction

LeCun's path towards autonomous machine intelligence [1] divides an agent into six modules: a configurator, perception, a world model, short-term memory, an actor, and a cost made of an intrinsic part and a critic. The world model is a joint-embedding predictive architecture (JEPA): it predicts the embedding of the next observation, not its pixels, and it is learned. Nothing in such a model marks which part of its latent state is a thing, a place or a rule. In an audit of one JEPA world model trained on a two-room world, the latents recovered the agent's position and local actions, but no individuated things, and the door modulated motion without blocking it [2].

This paper keeps LeCun's six modules with two departures. The world model is declared, not learned; and the configurator, which in LeCun (2022) sets up the other modules for the task, is a language model that writes the world model. The language model reads what perception reports about the scene and writes the rules of the world as a program. A deterministic engine runs the program: every proposed action goes through an acceptor that accepts or refuses it, with a reason, and computes the next state. The world model is then exact, it can be read, and a forbidden move is refused, not predicted unlikely. Perception reports labelled objects with their boxes, the ordinary output of current detectors and vision-language models. A JEPA can take such a scene graph as input, but it predicts in a latent space it learns, so its rules cannot be read; a declared world uses the scene graph directly, and every rule it applies is written down.

The question this paper answers comes before any planning: does a language model write the rules of the world it is shown, or the rules of the picture? A rule written from one scene can be *grounded*, bound to the objects and numbers in view (this door, these rows), or *lifted*, bound to kinds of objects (any door). Both are correct on the scene. Only the lifted rule holds when the layout changes. To isolate

the configurator, perception is held perfect: the scene graph is generated from the declared world, and each recovered world is scored against an exact answer key on 177,320 fixed proposals and on layouts never shown.

Six experiments vary one condition at a time.

- **E1** (one scene): every configurator writes a correct walker world, and every correct world is grounded. None of 21 holds on a layout with a second door.
- **E1b** (rules separated from facts, as a planning domain is separated from its problem, with a lint): every correct walker world is lifted and transfers, 11 of 11.
- **E1d** (one sentence asking for rules over kinds of objects): the two strongest models lift; the open-weight models do not.
- **E1c** (three layouts, no instruction): every world that compiles is lifted over the door, which varied, and grounded on the wall, which did not.
- **E1s** (a learner with only a simplicity prior, no language model): one scene gives a grounded rule at every cost of a constant; a second door gives a lifted rule at every cost.
- **E1w** (the converse: the wall varies, the door does not): every correct world of the wall arm lifts the wall rule, and 25 of 26 keep the door bound to one door; moving the door without adding one lifts nothing.

Three findings follow. Grounding is the default of every configurator tested. A rule is lifted over a factor of the scene when the observations separate the rules that bind it from those that do not, a condition we state for rules in Section 9 by analogy with the sufficient-variation condition for learned representations [13], or when the request asks for it. And because the world model is declared, the level at which a rule is bound is a property of the program, read from its text and checked before the world is used.

Section 2 describes the architecture and the declared world. Sections 3 to 8 report the six experiments, each with its own method, results and conclusion. Section 9 draws them together, and Section 10 concludes.

2. A Declared World Model in LeCun's Architecture

2.1. The Six Modules

The architecture keeps the six modules of LeCun (2022) and changes what fills them (Figure 1 and Table 1). It departs from LeCun (2022) in two ways. First, the world model is declared, not learned: in LeCun (2022) it is learned from observation, mostly self-supervised from video, and is meant to handle continuous states, uncertainty and several time scales, none of which the declared worlds of this paper exercise. Second, the configurator is a language model that writes the world model as rules, where in LeCun (2022) it modulates the parameters of the other modules for the task at hand. Perception outputs a scene graph, things and their positions, in place of a latent vector. Two of LeCun's design choices fit declared parts closely: the intrinsic cost is hard-wired and immutable, as red lines that refuse an action must be, and the critic is trained from experience, as rules mined from past journals are.

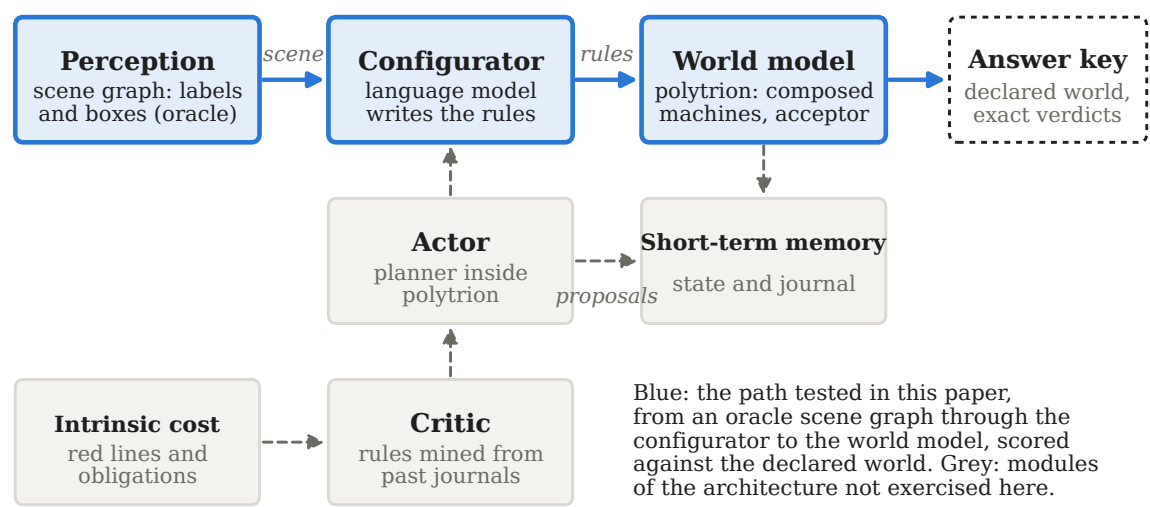


Figure 1. LeCun’s six modules filled by declared parts. The configurator writes the world model from what perception reports; the actor proposes actions that the world model’s acceptor accepts or refuses. This paper tests the blue path, with an oracle scene graph and the declared world as answer key.

Table 1. LeCun’s six modules and what fills them in the declared architecture. This paper tests the link from perception through the configurator to the world model.

Module	Role in LeCun (2022)	Filled here by
Configurator	Sets up the other modules for the task	A language model that writes the world’s rules, and revises a rule when the world disagrees (revision is not tested here)
Perception	Estimates the current state	A vision model that outputs a scene graph (oracle in this paper)
World model	Predicts the next state from an action; a learned JEPA	Polytrion: composed state machines with an acceptor; declared and exact
Short-term memory	Holds past, present and predicted states	Polytrion’s state and journal
Actor	Proposes actions optimized through the world model	A planner that searches inside polytrion
Intrinsic cost	Hard-wired drives	Red lines that refuse an action, obligations that add goals
Critic	Trained predictor of future cost	Rules mined from past journals

The declared world model takes the place of the JEPA’s predictor, and each JEPA component has a declared counterpart. The JEPA’s encoder maps an observation to an embedding; here perception maps it to a scene graph whose entries are named. The JEPA’s predictor maps an embedding and an action to the next embedding; here the acceptor maps a state and a proposed action to a verdict, a reason and the next state. The JEPA’s energy scores how compatible two embeddings are; here a refusal states that a transition is impossible, and why. Training the predictor becomes writing the rules, and it is done by the configurator from what it is shown. A learned predictor is identified only up to transformations of its latent space [12]; a declared rule is exact and can be read, so the question of what it is bound to can be answered from its text.

2.2. The World Model: Composed Machines with an Acceptor

The world model runs on polytrion, a deterministic simulation engine. A world is not one state machine: each thing (a walker, a door, a jug) is its own machine, and the relationships between things are the rules that compose them. Every action is a proposal. The acceptor, the one function that decides, either refuses the proposal with a reason from a fixed catalogue or accepts it and computes the next

state. The journal records every proposal, its decision, its reason and the states before and after. A plan is therefore checked against the world model before it is acted on, and the first proposal on which two worlds disagree is the address of their difference.

2.3. The Configurator and the Rule Schema

The configurator writes its rules as JSON in a fixed schema with four parts: *things*, picked from the scene by id or by label; *state* variables; *actions*, with parameters, guards and effects; and *refusals*, whose reasons come from the catalogue. A guard reads the state, the action’s parameters and the attributes of things, and refuses the action when its condition holds. Picking things by label, with a quantifier over every object of that label, allows a rule over a kind of object. An adapter interprets the JSON as polytrion machines; the model’s text is never executed as code.

The two forms of a wall rule, simplified to the cell a move reaches, show the distinction the paper turns on. A grounded rule copies the door from the scene:

```
{ "reason": "wall", "refuseIf": { "op": "and", "args": [
  { "op": "==", "args": [{ "var": "x1", "val": 32 }],
  { "op": "or", "args": [ { "op": "<", "args": [{ "var": "y1", "val": 13 }],
    { "op": ">", "args": [{ "var": "y1", "val": 18 } ] } ] } ] } }
```

A lifted rule reads the wall and quantifies over doors:

```
{ "reason": "wall", "refuseIf": { "op": "and", "args": [
  { "op": "==", "args": [{ "var": "x1", "val": { "attr": [ "wall", "bbox.xMin" ] } ] },
  { "op": "not", "args": [ { "any": "door", "as": "d", "where": { "op": "and", "args": [
    { "op": "<=", "args": [ { "attr": [ { "var": "d", "val": "bbox.yMin" } ], { "var": "y1", "val": 13 } ] },
    { "op": "<=", "args": [ { "var": "y1", "val": 18 }, { "attr": [ { "var": "d", "val": "bbox.yMax" } ] } ] } ] } ] } ] } }
```

Both agree on every proposal of the scene they were written from. On a layout with a second door, the first refuses the step through it and the second accepts it.

2.4. The Worlds

Two declared worlds serve as answer keys. In the two-room walker world, a walker moves on a 64-column grid split by a wall at $x = 32$, which it may cross only through a door spanning rows 13 to 18. In the hand-off jug world, jugs X and Y measure water from a tap with unlimited supply, jugs Z and T store it, and the acceptor refuses three kinds of action with a catalogue reason: emptying a jug, filling Z or T from the tap, and pouring out of Z or T. Each world is generated as a scene graph of labelled objects with bounding boxes and attributes (position, capacity, contents). Rules that no picture shows, such as that a store jug cannot be emptied, are given as a short text of stated constraints. The scene states what each object is, not what it allows: no relation says that the door connects the rooms. Figure 2 shows the walker layouts used in the paper: the scenes shown to the configurators, and the layouts never shown, on which transfer is tested.

2.5. STAR: The Four Parts a Declaration Names

Every declaration names four kinds of element: States, Things, Actions and Relationships (STAR). The categories come from an earlier study of the grammar a JEP world model learns [2], in which the latents recovered actions and places but no individuated things; a declaration names all four by construction. In this paper STAR serves as a codebook for errors: each recovered world that disagrees with the answer key is coded by the category of the first rule it gets wrong, with numbers counted among things and reported as Number, and errors of form (a rule the schema does not accept) coded None. The coding locates errors; the main measures, the binding of each rule and the layouts on which it holds, are read from the rule text and the scorer.

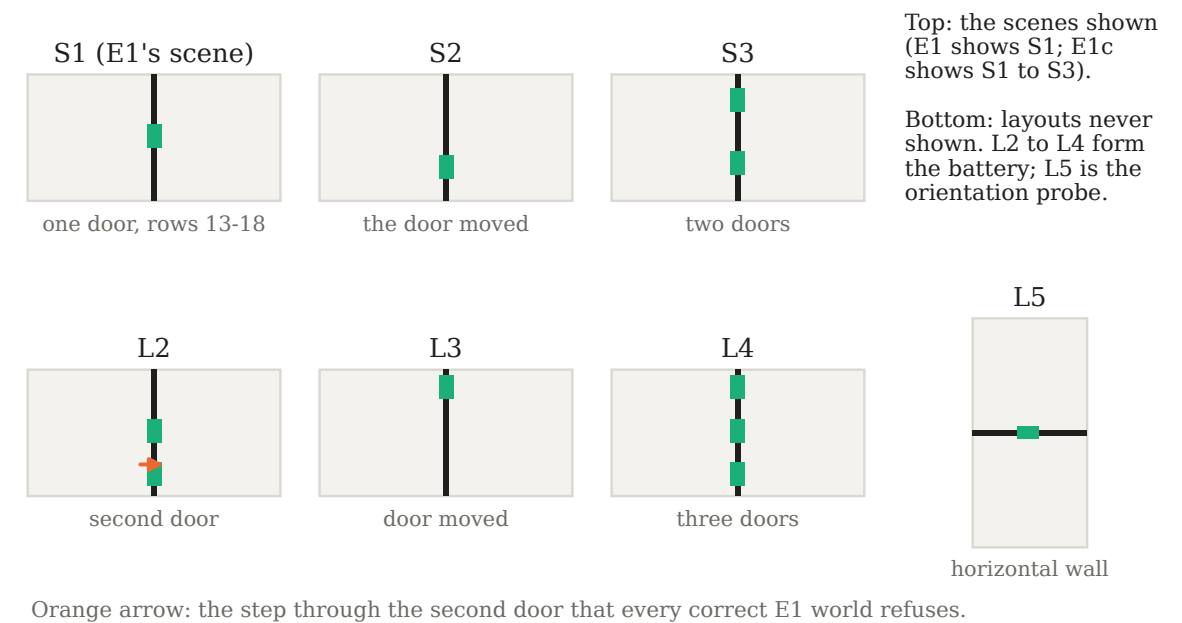


Figure 2. Walker layouts, 64 by 32 cells, wall in black, doors in green. S1 is the scene of E1, E1b and E1d; E1c and E1s show S1 to S3. L2 is the transfer layout of E1 and E1b; L2 to L4 form the battery of E1d and E1c; L5 turns the wall to the other axis and probes whether the wall rule reads the wall’s orientation.

2.6. Scope of This Paper

This paper tests one link of the architecture: from a perfect scene graph, through the configurator, to the declared world model. The actor, the intrinsic cost and the critic are not exercised, and perception is an oracle. Every experiment is declared before it runs: the question, the test set, the measures and the expectations are frozen in the repository, and every raw reply is committed.

3. Rules Written from a Labelled Scene

The first experiment (E1) asks whether a language model, given a scene graph that is exactly right, writes the rules of the world the scene shows. Perception is held perfect, so every error belongs to the configurator. A second question follows from the first: does a rule written from one scene still hold when the layout changes?

3.1. Method

Worlds and scenes.

The walker and jug worlds of Section 2 serve as answer keys. A third world, the same walker with a second door spanning rows 24 to 29, is never shown to the model; it tests transfer. A script generates each scene graph from the declared world files, so the two cannot disagree, and each declared rule is tagged as readable from the scene or from the text of stated constraints.

Rule schema.

The configurator writes the world in the schema of Section 2. Both declared worlds were written in the schema by hand, and the adapter reproduces them on every proposal, so the schema can express every declared rule, the door as a kind of object included.

Test set and scoring.

The test set holds 177,320 proposals, fixed before any run (seed 20261009): every reachable state of four jug instances crossed with 24 actions (175,200); every cell within three of the wall plus 64 random cells, crossed with the four unit moves (1,048); and the same construction on the two-door layout (1,072). Each proposal runs through the declared world and the recovered world. A recovered world is

correct when the two agree on every proposal in verdict, refusal reason and next state. A file that does not compile counts as incorrect. The adapter stops a proposal after 20,000 evaluations; no recovered world reached that bound.

Configurators.

Two open-weight models ran on Kaggle (two T4 GPUs, llama.cpp b6500): gpt-oss-20b and Qwen3-30B-A3B-Thinking-2507, 8 runs per world each, with up to 44,000 output tokens. Claude Sonnet 5.5 ran through the API, 4 runs per world, with adaptive thinking at high effort and the same prompts. The declaration fixed in advance that Claude Opus 5.5 would run only if Sonnet 5.5 fell below the gate; it ran afterwards as a declared supplement, with its question and expected answer stated before the run, and it does not enter the gate reading.

Counts.

Each result is a count of runs, and the runs are few. Where a count is compared, we give its exact 95% (Clopper–Pearson) interval. With four runs, 4 of 4 has an interval of 40 to 100% and 3 of 4 one of 19 to 99%, so the two cannot be told apart; pooled counts are tighter (0 of 21 gives 0 to 16%, 11 of 11 gives 72 to 100%). Differences between configurators within one experiment are reported as observed, not as estimates of a rate.

Gate.

Phase 1 passes when, for each world, at least one configurator writes a correct world in at least half of its runs. Transfer is reported but is not part of the gate. The declaration, its four amendments and every raw reply are committed in the repository, each amendment before the runs it concerns.

3.2. Every Configurator Recovers the Walker World

Table 2 gives the correct worlds. All four configurators write a correct walker world in at least three quarters of its runs, and phase 1 passes on both worlds: gpt-oss-20b reaches exactly half on the jugs, Sonnet 5.5 all four. No reply was truncated or refused.

Table 2. Correct worlds written from the oracle scene graph. Transfer counts the correct walker worlds that also hold on the two-door layout. Opus 5.5 is a declared supplement outside the gate.

Configurator	Runs per world	Walker	Jugs	Transfer
gpt-oss-20b	8	7	4	0 of 7
Qwen3-30B-A3B	8	7	1	0 of 7
Claude Sonnet 5.5	4	3	4	0 of 3
Claude Opus 5.5 (supplement)	4	4	4	0 of 4

3.3. Every Walker World Is Bound to the Door in View

The 21 correct walker worlds (17 in E1 proper, 4 from the Opus supplement) are correct for the scene they were written from. One scene cannot show whether its door is the only door or one of a kind of object, so both readings fit the evidence the configurator was given. The two-door layout tells them apart. Run on it, every one of the 21 worlds refuses the same step, from (31, 24) east through the second door, which the declared world accepts. The open-weight worlds still agree with the declared world on 98% of the transfer proposals: the difference is narrow, and it lies in what the wall rule is written over (Table 3).

Table 3. What the wall rule of each correct walker world is written over.

Written over	gpt-oss-20b	Qwen3-30B-A3B	Sonnet 5.5	Opus 5.5	All
Constants from the scene (32, 13, 18)	5	6	0	0	11
One door, by its id	2	1	3	4	10
The door type (any door)	0	0	0	0	0

The open-weight models mostly copy the door's coordinates into the guard as numbers; one gpt-oss world even declares the door as a thing by its label and then never uses it. Sonnet 5.5 and Opus 5.5 never copy numbers: they read the door's rows from the object door-1, and three Opus worlds go further and compute where a diagonal move crosses the wall, so that the move is checked against the door. The binding moves from numbers to a named object, and stops there. No configurator writes the rule a two-door world needs: a step across the wall is allowed when it passes through any object labelled *door*. The schema accepts that rule, and the hand-written reference world uses it and transfers.

In the terms of automated planning, the configurators write *grounded* rules, bound to the objects and numbers in view, where the task calls for *lifted* rules, bound to types. Each recovered walker world describes the scene it was written from, not the kind of world the scene is an instance of.

3.4. Most Jug Failures Are Errors of Form

The jug world separates the configurators. Sonnet 5.5 and Opus 5.5 write all eight of their jug worlds correctly. The open-weight models fail more often, and mostly in the form of the rule rather than its content. Across both worlds, of the 13 open-weight episodes that are not correct worlds, ten carry an error of form: a rule that says the right thing in a way the schema does not accept, such as a bare list where the schema expects an expression, or an attribute that does not exist. Three carry an error of content: a fill action with no guard, which lets the tap fill a store jug; refusals checked in an order that reports a different reason; and a walker world with no door. The fill with no guard is the most common error in schemas generated from text descriptions as well [5]: a missing precondition.

In an analysis chosen after the errors were seen, and so exploratory, two mechanical fixes of form (a bare list rewritten as the schema's list expression, a misplaced key rewritten as an operator) were applied to the eight files that did not compile. Three become correct worlds, two compile and then show the guard-order error, and three have errors of form outside the two fixes. The gate reading does not use this analysis.

3.5. The STAR Codes Locate the Errors in Relationships

Each failure was coded with the STAR codebook of Section 2: a rule-based first pass, adjudication of every unit, and a blind second coder who saw the codebook and the items but no code. Of the 32 units of E1 proper (the 14 episodes that are not correct worlds and 18 transfer failures), 21 are adjudicated Relationship, ten carry no STAR error (errors of form), and one is Action (refusal order). The first pass had coded 20 units as Number, because their first disagreeing cell lies next to a door edge; no bound was wrong in any of them. A sample of five adjudicated units, drawn with a recorded seed, was checked by the author, who agreed with all five.

Agreement with the blind coder is moderate on the world rows (Cohen's $\kappa = 0.41$, 14 units). On the fixed-door units the two coders split between Relationship and Action, and a second blind coder, on the Opus units, chose Action where the first had chosen Relationship. The codebook does not settle whether a rule bound to one door errs in the relation between wall and door or in the guard of the move. The finding does not depend on that choice: Table 3 is read from the rule text, not coded. A category for rules bound to an instance rather than a type is a candidate for the next codebook.

3.6. Grounded Rules from Scenes Are Not Reported in Prior Work

We found no prior measurement of whether a rule written from a single scene is lifted or grounded. Where language models write lifted rules, the formalism or a text description of the domain asks for them. Planning domains generated in PDDL are lifted because PDDL parameters are typed variables, and the rules come from a natural-language description of the domain [3–5]. When the input is an image, the scene yields grounded facts while the rules come from elsewhere: a given domain [6] or a text description refined by simulation [7]. Code world models transfer to new game levels [8] and name their laws by entity type [9], but they do not analyse the binding level, and rules learned under a prompt that asks for general rules still name specific items [10]. When the task is induction from

several examples, large models approach a dedicated inductive logic programming system [11], so lifting is within their reach.

E1 adds the missing case: one labelled scene, a schema that allows rules over types, and no formalism that requires them. Under those conditions four configurators, from a 20-billion-parameter open model to Opus 5.5, write grounded rules. The prior work predicts that lifting is induced from outside the model, by the formalism. The next experiment tests that prediction on the same scenes, with a schema that separates the rules, written over types, from the facts, the objects in view, as PDDL separates a domain from a problem.

3.7. Conclusion of E1

From a scene graph that is exactly right, language models write the rules of the world it shows: every configurator recovers the walker world, and the stronger models recover the jug world in every run. They write those rules for the scene, not for the world. All 21 correct walker worlds bind the wall rule to the door in view, by its coordinates or by its name, and none to the kind of object a door is; on a layout with a second door, every one refuses the step through it. A configurator that writes grounded rules gives a world model that is exact for the picture it was given and wrong for the next one. The declared world makes this visible as a property of the rule, at the first proposal that reaches the second door.

4. Rules Separated from Facts

E1 left one question open: is the grounding of the walker rules a limit of the configurators, or of the task as posed? Planning formalisms suggest the second. PDDL separates a domain of rules over typed variables from a problem of facts about objects, and language models that write PDDL domains write lifted rules [3,5]. The second experiment (E1b) adds this separation to E1 and changes nothing else.

4.1. Method

E1b keeps the scenes, the stated constraints, the worlds, the test set, the answer key, the scorer, the configurators and their settings. Only the output changes. The configurator writes a *domain*, the rules of the world written over labels such as *door*, *wall* and *jug*, and a *problem*, the objects in view with their ids, labels and boxes. A lint checks the domain before it is compiled and refuses three things: an object id, a number other than 0, 1 and -1 , and a label read outside a quantifier (“the door” rather than “any door”). The prompt states all three rules. A refused domain is scored as its own outcome, *rejected by lint*, since a configurator that can write only grounded rules shows it there.

Transfer now tests the domain alone: the model’s walker domain is compiled, unchanged, with the two-door problem generated by the oracle. Both reference worlds were rewritten as a domain and a problem under the lint; each agrees with its declared world on every proposal, and the walker domain also agrees on the two-door layout. The jug rules for the tap (which jugs it fills) can pass the lint only through the relation “a tap is above the jug”, since the stated constraints name Z and T; the jug results therefore include that inference.

The gate is set on transfer: E1b passes when at least one configurator writes a correct walker world that also transfers in at least half of its walker runs. The jug world checks that the separation does no harm. As in E1, Opus 5.5 runs as a declared supplement outside the gate.

4.2. Every Correct Walker World Transfers

E1b passes on Sonnet 5.5, whose four walker worlds are correct and transfer (Table 4). Every correct walker world of every configurator now transfers: 11 of 11 (95% interval 72 to 100%), against 0 of 21 in E1 (0 to 16%). Each is compiled unchanged with the two-door problem and agrees with the declared world on all 1,072 transfer proposals.

Table 4. Correct worlds in E1b, with E1 for comparison. Transfer counts the correct walker worlds that also hold on the two-door layout.

Configurator	E1b			E1		
	Walker	Transfer	Jugs	Walker	Transfer	Jugs
gpt-oss-20b (8 runs)	3	3 of 3	1	7	0 of 7	4
Qwen3-30B-A3B (8 runs)	0	none	0	7	0 of 7	1
Sonnet 5.5 (4 runs)	4	4 of 4	3	3	0 of 3	4
Opus 5.5 (4 runs, supplement)	4	4 of 4	4	4	0 of 4	4

The rules the configurators write explain the change (Table 5). Sonnet 5.5 and Opus 5.5 read the door through a quantifier in every walker world, and gpt-oss does so in four of eight. A step across the wall is allowed when it passes through any object labelled *door*; with a second door in the problem, the same rule admits it.

Table 5. What the wall rule of each E1b walker world is written over (all 24 walker worlds).

Written over	gpt-oss-20b	Qwen3-30B-A3B	Sonnet 5.5	Opus 5.5
Any door (quantified)	4	0	4	4
The door as one object	0	7	0	0
Constants for the door	0	1	0	0
No door	4	0	0	0

4.3. *The Lint Filters Nothing for the Stronger Models*

E1b changes two things at once, the format and the lint, so the overall result cannot say which of them did the work. The lint counts separate them in part. Sonnet 5.5 and Opus 5.5 had no lint rejection in 16 domains: every domain passed on its first and only attempt, and every walker domain read the door through a quantifier. For these two models the lint filtered nothing. Their walker transfer went from 0 of 7 in E1 to 8 of 8 in E1b. The lift came either from the format or from the instruction the prompt carries with it; Section 5 separates the two. The open-weight models are different: the lint refused 12 of their 32 domains (gpt-oss 2, Qwen 10), five of them for constants and seven for a label read outside a quantifier.

Qwen shows a binding the lint did not anticipate. In seven of its eight walker worlds it wrote a quantifier where a single object is read, which means “the one door”. The schema gives that form no meaning and the world fails at run time, so no recovered world reached a right verdict through it; but the lint does not refuse it. The same form appears for the tap in four jug worlds. A rule that refuses a quantifier wherever a single object is read closes this gap and belongs in the next declaration.

4.4. *The Separation Costs the Small Models on the Jug World*

The jug check holds for the stronger models (Sonnet 5.5 three of four, Opus 5.5 four of four) and fails for gpt-oss, which falls from four correct jug worlds in E1 to one. Three of its jug replies reached the 44,000-token limit before giving an answer: writing a domain, a problem and the tap inference costs the smaller model more than it can spend. Sonnet 5.5’s one jug miss repeats the guard-order error of E1. One correct gpt-oss jug world names the tap as the action’s agent; the schema allows it and it is right with one tap, but it binds the rule to that tap.

4.5. *Coding the E1b Errors*

The 33 units (29 episodes that are not correct worlds and four transfer failures) are adjudicated Relationship 20 (rules bound to one door, one tap or to the roles of Z and T through a capacity, and walker worlds with no door), None 12 (seven compile errors, two slips of form flagged by the lint, three truncations) and Action 1 (guard order). A lint finding is coded by one convention: on a name the proposal binds (the agent, or an action’s parameter) it is a slip of form; on a name that picks out an object of the scene (a door, a wall, the tap, a constant) it is grounding. All twelve lint-refused units

follow it. A sample of five adjudicated units, drawn with a recorded seed, was checked by the author, who agreed with all five.

Agreement with the blind second coder is low ($\kappa = 0.155$, 33 units). The 20 disagreements fall into two groups. In 13 units the blind coder read a rule refused by the lint as an error of form; its items stated why the lint refused a domain but not that a refusal marks a rule tied to one object of the scene. In seven units it coded a guard on the wall, the door or the tap as Action, the reading already met in E1. As in E1, the transfer result does not depend on the codes: Table 5 is read from the rule text.

4.6. The Format as a Declared Split

The contrast between E1 and E1b has a counterpart in learned world models. A joint-embedding predictive model trained with LeJEPA recovers the latent state only up to a global orthogonal transformation [12], so a learned dimension can mix factors that stay the same across related observations with factors that change. SplitJEPA [13] separates the two into an invariant and a variant block by training on pairs that share the invariant factors, and proves the split is identified only under sufficient variation: across the pairs, every variant factor must change. On PushCube, with the camera rotated by 10 degrees from its training pose, a behaviour-cloning policy on the invariant block succeeds in 71.5% of episodes, against 48.0% on a pre-trained visual representation (R3M), 25.0% on LeJEPA and 12.8% on raw pixels; on the harder PickCube task the same policy reaches 30.3% [13, Table 2].

E1 is the case of zero variation. One scene shows one door in one place, so nothing in it distinguishes the rule (cross the wall through a door) from the layout (this door, these rows); grounded rules are the expected result, by analogy with the identifiability argument for learned representations. E1b supplies the split by declaration rather than by data: the domain is the invariant block, the problem the variant block, and the lint plays the part of the agreement constraint. Unlike the learned blocks, which are identified from data only up to an invertible map within each block, the declared blocks are exact and named, because the format imposes the split; the split is then only as correct as what the format puts in the domain, and the lint's escape (the door as one object) shows the burden is real. SplitJEPA's condition also gives a later experiment its prediction. E1c (Section 6) keeps E1's format and shows the configurator several layouts of the same world; if variation is what identifies the invariant, the rules should lift without the separation.

4.7. Conclusion of E1b

Asked for the rules and the facts separately, the configurators write the rules over kinds of objects. Every correct walker world now holds on a layout it never saw, 11 of 11 against 0 of 21 in E1, and the two strongest configurators wrote lifted rules in every run without a single domain refused by the lint. Grounding in E1 therefore follows from the task as posed, not from a limit of the stronger models: a single scene does not say which of its features are the world and which are the layout, and the request can say it instead. The smaller models gain less, and gpt-oss pays for the separation on the harder jug world. Two questions remain: whether the instruction alone does the work of the format, and whether variation alone, several scenes of the same world, identifies the rules.

5. An Instruction to Abstract

E1b changed the output format and, with it, the prompt, which states the rules of the lint. For Sonnet 5.5 and Opus 5.5 the lint refused nothing, so either the format or the instruction lifted their rules. The third experiment (E1d) keeps E1 exactly and adds one sentence to the prompt: *write each rule over kinds of objects (by their label), never over a particular object or a number read from the scene*. There is no domain, no problem and no lint. The prompt's worked example, kept from E1, still names its objects by id.

E1d also replaces the single transfer layout with a battery of three layouts, none shown to any model: a second door on the same wall (L2, the transfer layout of E1 and E1b), the one door moved to other rows (L3), and three doors (L4). A fourth layout turns the wall to the other axis (L5); it probes a second grounding, the orientation of the wall, and is reported apart. The battery was applied once to

every correct walker world of all three experiments. The gate is E1b’s: at least one configurator must write a correct walker world that transfers to L2 in at least half of its walker runs.

5.1. The Instruction Lifts the Claude Models and Not the Open-Weight Models

E1d passes on Sonnet 5.5, three of whose four walker worlds transfer. Table 6 sets the three experiments side by side. For the two Claude models the sentence gives the same result as the format, within what four runs can distinguish: Sonnet 5.5 transfers in three runs of four (four with the format), Opus 5.5 in four of four (four with the format). Pooled, the two models transfer in 7 of 8 correct worlds with the sentence (95% interval 47 to 100%) and 8 of 8 with the format. Sonnet’s one miss reads the door from door-1 by id. For the open-weight models the sentence has no visible effect. None of their 12 correct walker worlds transfers (0 to 26%), and their rules are written over constants or door-1, as in E1. The three lifted worlds gpt-oss wrote in E1b therefore came from the format and its lint, not from the instruction it carries.

Table 6. Correct walker worlds that also transfer to the two-door layout, out of the correct walker worlds, in the three experiments. E1 adds nothing to the task; E1d adds one sentence; E1b adds the domain and problem format with its lint.

Configurator	E1	E1d (sentence)	E1b (format)
gpt-oss-20b	0 of 7	0 of 7	3 of 3
Qwen3-30B-A3B	0 of 7	0 of 5	none correct
Sonnet 5.5	0 of 3	3 of 4	4 of 4
Opus 5.5 (supplement)	0 of 4	4 of 4	4 of 4

The three experiments sort the configurators into three groups. Sonnet 5.5 and Opus 5.5 write lifted rules as soon as they are asked to. gpt-oss-20b writes them only when the interface separates the rules from the facts and refuses a grounded domain. Qwen3-30B-A3B writes them under neither condition. Grounding is the default in every case, and what it takes to leave it depends on the configurator.

5.2. The Binding Predicts the Battery

The battery confirms that the binding read from the rule text determines where a world holds (Table 7). Every world written over any door passes all three layouts. Every world that reads the door’s rows from door-1 passes only L3, where the one door moves, and fails wherever a second door appears. Every world written over constants passes none. Applied to the earlier experiments, the battery gives the same picture: 10 of the 21 correct E1 worlds pass L3, exactly those that read door-1, and none passes L2 or L4; all 11 correct E1b worlds pass all three. The L2 column agrees with the gate’s transfer test on all 32 worlds of E1 and E1b.

Table 7. Battery score (layouts passed out of L2, L3 and L4) by what the wall rule is written over, over the 52 correct walker worlds of E1, E1b and E1d.

Written over	Layouts passed	Holds when
Any door (quantified)	3 of 3	doors are added or moved
door-1, rows read from the object	1 of 3	the one door moves (L3)
Constants from the scene	0 of 3	never

5.3. A Lifted Door Does Not Make a Lifted Wall

The orientation probe asks a further question of the rules that pass the battery: is the wall itself written as a kind of object? Of the 52 correct walker worlds, 19 read the wall as a column (its horizontal extent only), 19 never read the wall and use a constant for its column, 13 read it as a region (all four edges of its box, with the target cell tested inside the box and outside every door). One other world reads the wall in none of these ways. No column or constant world survives a horizontal wall. Seven

region worlds do (three from gpt-oss in E1b, four from Opus 5.5 in E1b and E1d); the other six still compare a single coordinate in another clause. The test set proposes unit moves only, so a pass shows that the wall was written as a region, not that the rule reads the wall's axis. Lifting proceeds object by object: a configurator can quantify over doors and still take the wall's orientation from the one scene it was shown.

5.4. Coding the E1d Errors

The 30 units (16 episodes that are not correct worlds and 14 transfer failures) are adjudicated Relationship 15 (a wall rule fixed to the door in view), None 9 (errors of form and truncations) and Action 6 (the guard-order error of E1). Agreement with the blind second coder is high on the world rows ($\kappa = 0.88$, 16 units) and lower overall ($\kappa = 0.35$, 30 units). All 15 disagreements concern one question, whether a wall rule fixed to one door is a Relationship or an Action error, and the blind coder chose Action each time. A sample of five adjudicated units was checked by the author, who agreed with all five. As in E1 and E1b, the transfer result rests on the binding read from the rule text, not on the codes.

5.5. Conclusion of E1d

One sentence asking for rules over kinds of objects lifts the rules of the two strongest configurators as the separate format does, within what four runs per model can distinguish, and does not lift the rules of the open-weight models at all. Grounding is the default for every configurator; leaving it takes an instruction for the strongest and an interface that enforces it for a mid-size model. The battery shows that the binding read from the rule text predicts exactly where a recovered world holds, and the orientation probe shows that lifting is partial: a rule can quantify over doors and still fix the wall it was shown. A declared world makes each of these levels visible as a property of the rule, before the world is used.

6. Variation without Instruction

E1b and E1d lifted the rules by asking for abstraction, through a format or through one sentence. SplitJEPa's condition suggests a third route: a rule is separated from the layout only when the layout varies across the observations [13]. The fourth experiment (E1c) tests that route. It keeps E1's format and prompt, with no lint and no instruction to abstract, and shows the walker world in three layouts instead of one.

6.1. Method

The prompt shows three labelled scene graphs of the walker world and asks for one set of rules that is correct in all three. The wall is at the same column in every scene. Scene S1 is E1's scene, with one door; S2 moves the one door to other rows; S3 has two doors. The ids are stable: door-1 is the first door of every scene and door-2 exists only in S3. What varies across the scenes is the position and the number of doors. The wall's position and orientation do not vary.

A recovered world is correct when it agrees with the declared world on the test sets of all three scenes. The battery and the probe are those of E1d, and none of their layouts is among the scenes shown. The gate is that at least one configurator writes a correct world that transfers to L2 in at least half of its runs. The configurators are gpt-oss-20b and Qwen3-30B-A3B (8 runs each) and Sonnet 5.5 (4 runs). Opus 5.5 was to run only if Sonnet 5.5 fell below the gate; it did not run.

6.2. Every World That Compiles Is Lifted

E1c passes on Sonnet 5.5, whose four worlds are correct and transfer. No world that compiled is wrong: all 7 that compiled are correct on the three scenes, and all 7 pass the battery 3 of 3 (Table 8). The door is read through a quantifier in every one of them. No correct world is written over constants or over door-1, since S3's second door rules both out.

Table 8. Correct walker worlds in the four experiments, with in brackets those that also transfer to the two-door layout. E1c shows three scenes and asks for nothing else.

Configurator	E1	E1d (sentence)	E1b (format)	E1c (three scenes)
gpt-oss-20b (8 runs)	7 (0)	7 (0)	3 (3)	2 (2)
Qwen3-30B-A3B (8 runs)	7 (0)	5 (0)	0 (0)	1 (1)
Sonnet 5.5 (4 runs)	3 (0)	4 (3)	4 (4)	4 (4)
Opus 5.5 (4 runs, supplement)	4 (0)	4 (4)	4 (4)	not run

Variation is the first condition that lifts Qwen3-30B-A3B. Its one correct world is also its first lifted world in the four experiments; neither the sentence nor the format produced one. For the open-weight models the lift comes with fewer correct worlds: 3 of their 16 runs give a correct world (95% interval 4 to 46%), against 12 of 16 in E1d (48 to 93%), but all 3 transfer, against none of the 12.

6.3. *The Bound Variable Is Where Expression Breaks*

Eleven of the 16 open-weight replies do not compile, and two gpt-oss replies used the whole output limit in reasoning and gave no answer. All eleven compile errors contain the same slip. The rule quantifies over doors, as three scenes require, and then refers to the quantified door by a bare name where the schema expects a variable reference. Four of the eleven carry a second slip of the same kind, such as a label or the agent used without being declared. The open-weight models therefore reach the lifted rule and fail to write it in the schema. Under variation the difficulty moves from what the rule is about to how it is written.

All 13 units are adjudicated None (errors of form and truncations), under the convention of E1b extended to quantified variables: a name the rule’s own quantifier binds is a bound name, whatever its spelling. The blind second coder agrees on 10 of 13 units. Kappa is not informative here, since every adjudicated code falls in one category. The three disagreements are names used without being declared, which the blind coder read as errors of Thing. A sample of five adjudicated units was checked by the author, who agreed with all five.

6.4. *The Wall Stays Grounded Where It Never Varied*

The wall is at the same column in all three scenes, so variation gives no reason to lift it. The rule texts show that it was not lifted (Table 9). In 17 of the 18 worlds that have a text, the wall is a column: a constant in 11, a value read from the wall’s horizontal edge in 6. Both forms assume a vertical wall. Only one world reads the wall as a region. None of the 7 correct worlds passes the probe, the region reader included. Among the worlds that did not compile the wall is a constant more often (10 of 11) than among the correct ones (1 of 7); the groups are small and not independent of the slip that stops them compiling, so the difference is not read as an effect.

Table 9. How the door and the wall are written in the 18 E1c walker worlds that have a text (the 7 correct and the 11 that do not compile).

Written as	Correct	Did not compile	All
Door: a quantifier over the label	7	11	18
Wall: a constant column	1	10	11
Wall: the column read from the wall	5	1	6
Wall: a region (all four edges read)	1	0	1

The result matches SplitJEPA’s condition, read by analogy for rules. The factor that varied, the door, is lifted in every world. The factor that did not vary, the wall’s position and orientation, stays grounded in nearly every world and in every correct one.

6.5. Limits of E1c

The prompt says that the three scenes are one world. This statement is part of the variation condition, not an addition to it: SplitJEPA's training pairs carry the same information, since a pair is known to share its invariant factors, and observations not known to come from one world identify no invariant, for a learned or a declared model. The runs are few, and the open-weight worlds that compiled number three. The probe uses unit moves only, as in E1d, so a failure shows that the wall rule assumes a vertical wall, and a pass would not show that a rule reads the wall's orientation. The wall table for the worlds that do not compile is read from their text, since they cannot be run.

6.6. Conclusion of E1c

Three layouts of the same world, with no instruction to abstract and no separate format, lift the door rule in every world that compiles, for all three configurators, including the one that neither the sentence nor the format had lifted. The wall, which never varied, stays grounded. Lifting follows variation factor by factor, in line with SplitJEPA's condition for learned representations. For the open-weight models the limit then lies in expression: they reach the quantified rule and fail to write its variable in the schema. Section 7 asks whether a learner with only a simplicity prior shows the same pattern.

7. A Simplicity Learner on the Same Scenes

A soft simplicity bias offers an account of the shift from E1 to E1c [14]: among the rules consistent with the observations, the shortest is preferred, and which rule is shortest depends on what the observations show. The fifth experiment (E1s) tests this account without a language model. A deterministic learner returns the shortest wall rule consistent with labelled transitions from the E1c scenes, and the battery and the probe are applied to what it returns.

7.1. Method

The hypothesis space is the wall guard of the walker world, written in a restricted grammar of the E1 schema: comparisons of a coordinate of the target cell against a value, combined with and, or and not. A value is an integer constant, an attribute of an object named by id, or an attribute of a variable bound by a quantifier over a label. The prior chooses uniformly at every expansion of the grammar, and the description length of a rule is minus the base-2 logarithm of its prior probability. The learner enumerates rules in order of increasing description length, up to 52 bits, and returns every consistent rule at the minimum. Its data are the labelled transitions of the scenes given, built as E1's test sets are, with the declared world's verdicts. The language models never see such data.

The learner runs on four sets of scenes: S1 alone (C1, the E1 condition), S1 and S2 (C12, the one door moved), S1 and S3 (C13, a second door) and all three (C123, the E1c condition). The declared prior gives an integer constant 6 bits, the cost of one value among the 64 columns of the grid. Because the cost of a constant against the cost of reading an object decides between grounded forms, an exploratory sweep varies it from 1 to 6 bits, with everything else unchanged. A returned rule is classified by its meaning: it is lifted when it is correct on S3 and passes L2 to L4.

7.2. One Scene Gives a Grounded Rule at Every Cost

The declared expectation for C1, a winner over constants, failed: at 6 bits per constant the shortest consistent rule reads the door from a quantifier (36.2 bits, against 39.8 for door-1 by id and 42.2 for constants). The rule it writes is the binding Qwen3-30B-A3B wrote in E1b, the door as one object: some door whose rows exclude the target cell. It is correct for one door and wrong for two. It fails S3 on 44 of 1,072 proposals and passes only L3 in the battery.

Figure 3 gives the winner of each condition across the sweep. On one scene the winner is grounded at every cost: it copies constants when a constant costs 3 bits or less, and binds the one door above. The constants rule costs $25.4 + 3c$ bits for a constant cost of c bits, so the switch lies near 3.6 bits. The

cost of a constant decides the form of the grounding, never whether the rule is grounded. Door-1 by id never wins, although at 3 bits it is within 0.02 bits of the winner.

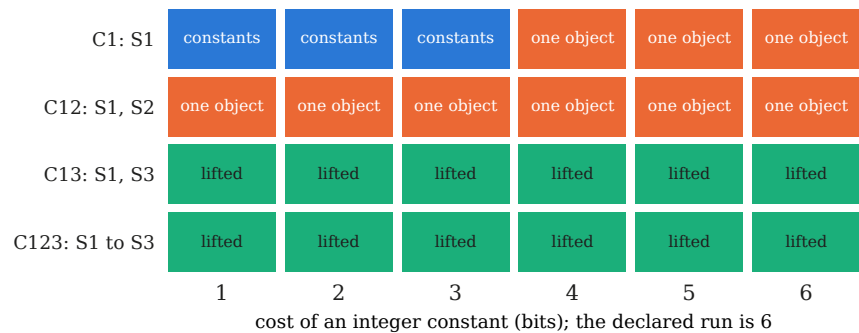


Figure 3. Binding of the shortest consistent wall rule by set of scenes and cost of a constant. Constants: the door’s rows copied as numbers. One object: some door whose rows exclude the cell, correct for one door and wrong for two. Lifted: correct on S3 and passes L2 to L4.

7.3. A Second Door Lifts the Rule, a Moved Door Does Not

Moving the one door (C12) rules out constants but does not lift the rule: the winner is the door as one object at every cost. A second door (C13 and C123) gives a lifted winner at every cost, correct on all three scenes and passing the battery 3 of 3. The factor that must vary is the number of doors, not their position. This is the analogue for rules of SplitJEPa’s condition, applied factor by factor: the rule is lifted over the factor that varies across the observations, and only over that factor.

The wall shows the same condition from the other side. No returned rule reads the wall’s own box. Above 3 bits the winners take the wall’s column from the door’s left edge; at 3 bits or less they write it as the constant 32, with one exception (C12 at 3 bits, which reads the door’s edge). Every returned rule fails the probe. The wall never varies in the scenes, so for the learner it is not a factor at all.

7.4. The Learner and the Language Models

The learner reproduces the grounded forms of E1 and the factor-wise lift of E1c from labelled transitions and a simplicity prior. It does not account for two results. The language models reach their rules from the scene alone, with no labelled transitions. And in E1d the Claude models, given one scene and asked to abstract, wrote the lifted rule, which the learner never returns from one scene at any cost. Their own rules also go against length: across the 59 correct walker worlds of E1, E1b, E1d and E1c, the median wall rule, counted in nodes of its JSON tree, has 52 nodes over constants, 109 over door-1 and 142 over any door. The instruction acts on what the model takes the rule to be, not on how short it is.

7.5. Limits of E1s

The grammar is restricted to comparisons of a target coordinate against a value; the unrestricted grammar was too large to enumerate. The shortest rules of each condition at the declared prior were seen in dry runs before the declaration was frozen, and the expectations were left as written. At 1 bit per constant the search stops at 38 bits, below the lifted winner, so it cannot exclude a grounded rule between 38 and 40.5 bits on C13 and C123. A free cost per constant is not a probability: the sweep shows where the answer changes, not what the prior should be.

7.6. Conclusion of E1s

A learner with only a simplicity prior writes a grounded wall rule from one scene at every cost of a constant, and a lifted one as soon as a second door is shown. The cost of a constant decides whether the grounding copies numbers or binds the one door, the two forms the configurators wrote in E1. Moving the door does not lift the rule; adding a door does. A simplicity prior therefore explains why variation lifts the rules, and which variation does. It does not explain the instruction effect of E1d, where the strongest models wrote the lifted rule from one scene because they were asked to.

8. Varying the Wall

E1c and E1s found that variation lifts a rule over the factor that varies and only over it, but tested the claim in one direction: the doors varied and were lifted, the wall did not and stayed grounded. The sixth experiment (E1w) tests the converse and one control, as E1c with other scenes. In arm W the wall varies and the door does not: one door, always spanning the same six cells along the wall, with the wall at column 32 (W1, the scene of E1), at column 20 (W2) and horizontal (W3). In arm D the scenes are S1 and S2 of E1c: one door, moved, with the wall fixed. Neither arm ever shows two doors, so by the condition of Section 9 neither identifies a rule over the number of doors; a world that holds on two or three doors lifted that factor without evidence, from the model’s prior.

Two layouts never shown test the wall: WL1 moves it to column 44, WL2 makes it horizontal at another row. The door battery (L2 to L4) tests the door, as before. The configurators are those of E1c, with 8 runs per arm for each open-weight model and 4 for Sonnet 5.5; Opus 5.5 was to run only where Sonnet 5.5 wrote fewer than 2 correct worlds, and did not run. The E1s learner, with its grammar extended by a choice of the wall’s axis, ran on W1, on W1 and W2, and on all three. Its description lengths are not comparable with those of E1s, since the grammar differs.

8.1. The Wall Is Lifted When It Varies

All 12 correct worlds of arm W read the wall from the wall object and pass both WL1 and WL2; none writes the wall as a constant column (Table 10). The open-weight models read the wall on both axes in all 8 of their correct worlds; Sonnet 5.5 reads both axes in two worlds and tests the wall’s axis in the other two. These are the two open-weight models that the request of E1d did not lift at all: varying the wall lifted its rule in every one of their correct worlds.

Table 10. Correct worlds in E1w and the layouts they hold on. WL1 and WL2 move and turn the wall; L2 to L4 add or move doors. A correct world that passes L2 and L4 lifted over the number of doors, which neither arm varies.

Configurator	Arm W (wall varies)		Arm D (door moved)	
	Correct	WL1, WL2 / L2 to L4	Correct	L2 to L4
gpt-oss-20b (8 runs)	5	5 / 0	5	0
Qwen3-30B-A3B (8 runs)	3	3 / 0	5	0
Sonnet 5.5 (4 runs)	4	4 / 1	4	0

8.2. The Door Stays Bound Where It Does Not Vary

In arm W the door never changes, and 11 of the 12 correct worlds keep it bound to door-1 by id: they pass L3, where the one door moves, and fail L2 and L4. In arm D the door moves but stays one door, and all 14 correct worlds, from all three configurators, bind it to door-1 and pass only L3. The instruction to write one set of rules for all the scenes did not act as a request to abstract, for Sonnet 5.5 or for the open-weight models. In arm D the wall, which never varies, stays grounded in most open-weight worlds: a constant column in all 5 of Qwen’s correct worlds and in 2 of gpt-oss’s.

One Sonnet 5.5 world in arm W (1 of 4) writes the door as any door and passes the whole battery. The scenes gave it no reason to, so this is a lift from the prior. It is the only one among the 26 correct worlds of E1w.

8.3. The Learner Needs the Turned Wall

On W1 alone and on W1 and W2, the learner returns the same 53 rules tied at the minimum description length, and only one of them is correct on all three scenes; the others read the wall’s position along the x axis only. W2 is another vertical wall, so it cannot separate them. With W3 added, the learner returns exactly one rule, the correct one. It reads the wall’s position from the door’s position across the wall, never from the wall object, and keeps the door as one door (1 of 3 on the battery). The factor that must vary for the wall rule is the wall’s axis, as the number of doors was for the door rule in E1s.

8.4. Limits of E1w

The Sonnet 5.5 runs are four per arm, and the one lift from the prior is one world. Of the open-weight replies, 7 of 16 in arm W and 6 of 16 in arm D did not compile or were cut off, and are not analysed further. The layouts test unit moves only, so a pass on WL2 shows that a rule handles a horizontal wall at the cells a unit move reaches, not every move. The finer labels for how the wall is read are an analysis added after the declaration; the declared binding table is the measure.

8.5. Conclusion of E1w

Varying the wall lifts the wall rule, and holding the door fixed keeps the door rule bound, for every configurator: all 12 correct worlds of arm W pass the moved and turned wall, and 25 of the 26 correct worlds of both arms stay bound to one door. Moving the door without adding one does not lift the door rule, for Sonnet 5.5 as for the open-weight models. Together with E1c, the result holds in both directions: a configurator shown varied scenes lifts exactly the factors that vary, with one exception in 26 worlds where Sonnet 5.5 lifted the door unprompted.

9. Discussion

9.1. Grounding Is the Default

Given one scene, every configurator wrote grounded rules, and so did the simplicity learner at every cost of a constant (Sections 3 and 7). One scene does not say whether its door is the only door or one of a kind, so a grounded rule is a correct reading of the evidence, not an error of the model. The form of the grounding varies: the open-weight models copied the door’s numbers, the Claude models bound the door by its id, and the learner did one or the other depending on what a constant costs. What does not vary is that nothing in one scene identifies the lifted rule.

9.2. Three Conditions Lift the Rules

Three conditions moved the configurators from grounded to lifted rules (Figure 4). The format that separates rules from facts lifted every configurator that wrote a correct world. The request, one sentence, lifted the two strongest models only. Variation, three layouts with nothing asked, lifted every world that compiled, for all three configurators tested, including the one that neither the request nor the format had lifted. The open-weight models then failed at expression: they reached the quantified rule and wrote its variable in a form the schema does not accept.

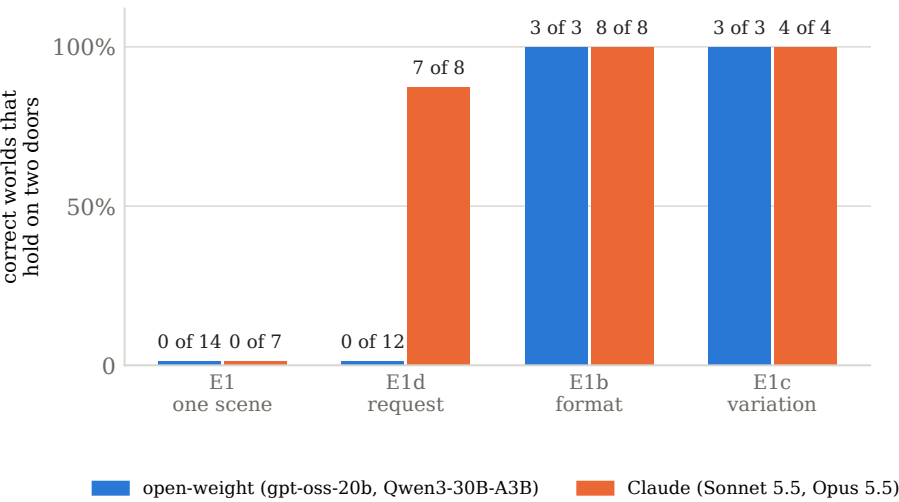


Figure 4. Correct walker worlds that hold on the two-door layout, out of correct walker worlds, by condition. Open-weight: gpt-oss-20b and Qwen3-30B-A3B; Claude: Sonnet 5.5 and Opus 5.5 (Opus 5.5 did not run in E1c). The counts are small; 3 of 3 has a 95% interval of 29 to 100%.

The format and the request supply by declaration what variation supplies by data. The domain plays the part of the invariant block of SplitJEPa, the problem the part of the variant block [13], and the sentence names the split without the format. Variation supplies it in the way SplitJEPa's condition describes: the observations are known to come from one world, and the factor to be separated changes across them.

9.3. Variation Acts Factor by Factor

Variation lifted only what varied. In E1c the door, which changed position and number across the scenes, was lifted in every world; the wall, which never moved, stayed grounded in 17 of 18 worlds and in every correct one, and no correct world survived a horizontal wall. The learner of E1s showed which variation counts: moving the one door did not lift its rule, and adding a second door did, at every cost of a constant. The factor that must vary is the one the rule quantifies over, here the number of doors. E1w tested the converse. When the wall varied and the door did not, every correct world of every configurator lifted the wall rule and passed a moved and a turned wall, while 25 of 26 kept the door bound to one door; moving the door without adding one lifted nothing, for Sonnet 5.5 as for the open-weight models. The learner again needed the factor itself to vary: a second vertical wall did not separate the axis-aware rule from the column rules, a horizontal wall did. This is an analogue, for rules, of the sufficient-variation condition of SplitJEPa [13], and the analogy has a limit. SplitJEPa identifies latent factors from observations; it does not promise that a function of those factors extrapolates to values never observed. What fails on a horizontal wall is extrapolation of this kind, not identification.

The condition can be stated for rules directly. Let H be the class of rules the schema can express, S the scenes shown with their labelled transitions, and f a factor of the layout (the number of doors, their rows, the wall's column, its axis). A rule is *lifted over f on S* when every rule of H consistent with S gives the same verdicts on every layout that differs from the scenes of S only in f . The scenes *vary f sufficiently* when they rule out every consistent rule that disagrees with the declared world on such a layout. For the door rule of E1s this is a property of the grammar, checkable by enumeration: S1 is consistent with both the one-door and the any-door rule, and so is S1 with S2, since both show one door; S3 rules the one-door rule out. A simplicity prior then chooses among the rules that remain, which is why the cost of a constant decides the form of the grounding but never whether the rule is grounded. A language model sees no labelled transitions, so the definition does not apply to it as such; what the experiments test is whether its rule is among those the scenes leave consistent, and which one it picks.

9.4. What a Simplicity Prior Explains

A simplicity prior [14] accounts for the effect of variation: on one scene a grounded rule is the shortest consistent rule, and once two doors are shown the lifted rule is. It does not account for the effect of the request. In E1d, with one scene, the Claude models wrote the lifted rule because they were asked to; the learner never returns it from one scene, and the models' own lifted rules are longer than their grounded ones. The request changes what the model takes the rule to be, not how short it is. A language model as configurator thus has two routes to a lifted rule, the data and the request, where a learned predictor has only the first.

9.5. Consequences for the Architecture

Three consequences follow for the declared architecture. First, the configurator should see the world under variation: several frames of the same world in which the factors to be lifted change, which an agent that moves through its world gathers on its own. Second, the interface should separate rules from facts, which costs the strongest models nothing and lifts gpt-oss-20b, which the request does not reach. Third, the binding of a rule can be checked before the world is used: it is read from the text, and a battery of layouts the configurator never saw confirms it. In every experiment, the binding read from the text predicted exactly where a world holds. The first proposal on which a world fails is

also a counterexample the configurator could use to revise its rule, which is the revision step of the architecture.

9.6. Limits

The scene graph is an oracle, exact by construction; real perception adds noise, and the grounded rules over constants are the ones it would affect first. The domain is small: one wall, one to three doors, unit moves, and a jug puzzle. The runs are few, four or eight per configurator and condition, so differences within one experiment are reported as observed and not as rates. The battery and the probe test unit moves only. The simplicity learner works in a restricted grammar, and its cost per constant is swept, not derived. The results concern the models and versions named, from two families (two open-weight models and two Claude models), run once each with the settings stated. The declared worlds are deterministic and discrete; a declared world model has yet to show how it represents uncertainty, partial observability and continuous dynamics such as contact and friction.

9.7. Future Work

The limits above set the order of the next studies.

- **Perception from pixels.** Replace the oracle with a detector and then a vision-language model on rendered images, measure the noise on the boxes, and test whether lifted rules survive it better than grounded ones.
- **A learned baseline.** Train a learned world model, a JEPA on rendered frames or an object-centric one, on the same layouts, and score it on the same battery and probe. This tests directly whether the declared model transfers where the learned one does not, and whether the learned model's failures can be located.
- **Planning.** Run the actor through the declared world model, with the acceptor refusing illegal rollouts, against a learned world model at equal planning compute.
- **Uncertainty and continuous dynamics.** State and test what a declared world cannot represent: probabilistic outcomes, partial observability, contact and friction, with Push-T as a first limit case.
- **A richer domain.** Several object types, several factors varying together, and properties that are quantities to infer, not labels.
- **Revision from counterexamples.** Give the configurator the first proposal on which its world fails and ask it to repair the rule, which turns the acceptor from an evaluator into a learning signal.
- **More configurators.** Models from other families, and grammar-constrained decoding for the open-weight models, so that their results concern rules and not syntax.

10. Conclusions

This paper fills LeCun's architecture with a declared world model: a language model, as configurator, writes the rules of the world from a scene graph, and a deterministic engine runs them, accepting or refusing each proposed action. Held to a perfect scene graph and an exact answer key, the configurators recover the rules of the scene they are shown, but bind them to the objects and numbers in view: from one scene, no correct walker world held on a layout with a second door. Three conditions lift the rules over kinds of objects. Separating rules from facts lifts every configurator that writes a correct world; asking for rules over kinds lifts the strongest models; showing the same world under several layouts lifts exactly the factors that vary, the door when the doors vary and the wall when the wall varies, in line with the sufficient-variation condition for learned representations. A learner with only a simplicity prior reproduces the effect of variation but not the effect of the request. Because the world model is declared, what each rule is bound to is a property of its text, and the layouts on which it holds are known before it is used. The next step replaces the oracle with real perception.

Acknowledgments: The open-weight configurators (gpt-oss-20b and Qwen3-30B-A3B-Thinking-2507) ran on GPUs provided free of charge by Kaggle. We thank Kaggle for this compute.

References

1. Y. LeCun. A path towards autonomous machine intelligence. OpenReview, version 0.9.2, 2022.
2. J.-J. Dubray. [Title of the JEPA audit paper]. Preprints.org, 2026. doi:10.20944/preprints202609.2310.v1.
3. L. Guan, K. Valmeekam, S. Sreedharan and S. Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. NeurIPS 2023.
4. J. Oswald, K. Srinivas, H. Kokel, J. Lee, M. Katz and S. Sohrabi. Large language models as planning domain generators. AAAI 2024 (student abstract); arXiv:2405.06650.
5. M. Tantakoun, G. Garey, C. Muise and X. Zhu. ALPSBench: can large language models reason their way through planning formalisms? ICAPS 2026 Workshop on Language Models for Planning.
6. K. Shirai et al. Vision-language interpreter for robot task planning. ICRA 2024.
7. Y. Hao, Y. Chen, C. Fan and Y. Zhang. Simulation to rules: a dual-VLM framework for formal visual planning. arXiv:2510.03182, 2025.
8. W. T. Piriyaakulkij, Y. Liang, H. Tang, A. Weller, M. Kryven and K. Ellis. PoE-World: compositional world modeling with products of programmatic experts. NeurIPS 2025.
9. Z. Khan, A. Prasad, E. Stengel-Eskin, J. Cho and M. Bansal. One life to learn: inferring symbolic world models for stochastic environments from unguided exploration. ICLR 2026.
10. S. Zhou, T. Zhou, Y. Yang, G. Long, D. Ye, J. Jiang and C. Zhang. WALL-E: world alignment by rule learning improves world model-based LLM agents. arXiv:2410.07484, 2024.
11. J. P. Gandarela, D. Carvalho and A. Freitas. Inductive learning of logical theories with LLMs: an expressivity-graded analysis. AAAI 2025.
12. When does LeJEPA learn a world model? arXiv:2605.26379, 2026.
13. R. Hua, Z. Liu, Z. Zhao and Y. Zheng. SplitJEPA: learning invariant and variant latent worlds without reconstruction. arXiv:2610.12349, 2026.
14. A. G. Wilson. Deep learning is not so mysterious or different. arXiv:2503.02113, 2025.

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.