Concept Paper

Re-Thinking Training Data: From Tokens and Parameters to Tasks and Capabilities

Feng Chen *

Posted Date: 4 December 2025

doi: 10.20944/preprints202512.0499.v1

Keywords: capability-centric training; capability graphs; task space coverage; curriculum learning; synthetic data generation

 doi:10.20944/preprints202512.0499.v1

Concept Paper

Re-Thinking Training Data: From Tokens and Parameters to Tasks and Capabilities

Feng Chen

University of Chinese Academy of Sciences; 1401761846@qq.com

Abstract

Current large language model (LLM) practice is organized around two primary axes: the number of tokens consumed during training and the number of parameters used to fit them. These proxies have served as effective scaling knobs, but they are only loosely coupled to what actually matters for scientific and real-world deployment: which *capabilities* the model acquires, how robustly it can exercise them, and how gracefully it fails. Training on undifferentiated text streams with aggregate loss functions treats "all tokens as equal," while downstream users care about highly structured task families such as multi-step reasoning, tool use, code synthesis, experimental planning, or negotiation. In this Perspective, we argue for a shift from token-centric to capability-centric training. We sketch a framework in which tasks are represented as nodes and edges in a *capability graph*, capturing pre-requisites, compositional structure, and transfer relations. This representation enables new training regimes that explicitly target underdeveloped regions of capability space through curriculum design, data selection, and synthetic data generation. It also suggests system-level architectures where different agents specialize on subgraphs, coordinated by a higher-level planner. We further discuss how capability-centric thinking reshapes evaluation, red-teaming, and safety: instead of aggregate benchmarks, we can reason about coverage, brittleness, and failure modes at the level of subgraphs and capability clusters. We conclude by outlining open challenges—constructing capability graphs from messy real-world interactions, maintaining them as models evolve, and preventing overfitting to the graph itself—and argue that the transition from tokens and parameters to tasks and capabilities will be central to the next phase of LLM research.

Keywords: capability-centric training; capability graphs; task space coverage; curriculum learning; synthetic data generation

1. Introduction

Over the past five years, large language models (LLMs) have become the dominant paradigm in natural language processing and, increasingly, in scientific and engineering workflows. Models such as GPT-3 and its successors are trained on trillions of tokens using hundreds of billions of parameters, and they exhibit striking few-shot and in-context learning abilities across a wide variety of benchmarks and tasks [1-6]. This empirical success has been closely associated with "scaling laws": predictable relationships between model size, dataset size, compute and aggregate loss, which have guided the design of successively larger models [2,3]. In parallel, alignment techniques such as reinforcement learning from human feedback (RLHF) have transformed these raw foundation models into instruction-following assistants that can interact with non-expert users [7].

Yet, we still largely talk about and optimize LLMs in the language of *tokens* and *parameters*. When a new model is proposed, it is typically described by its parameter count, pre-training token count, and aggregate training loss. Scaling curves and compute-optimality analyses treat the training process as minimizing a single scalar objective—cross-entropy over an undifferentiated text stream—subject to a budget [2,3]. This "token-centric" viewpoint has been essential for engineering progress,

 doi:10.20944/preprints202512.0499.v1

but it obscures a crucial question: *what capabilities are actually being learned, in what order, and at what level of robustness?*

Empirically, capabilities do not grow smoothly with these low-level quantities. As models scale, they exhibit so-called emergent abilities: qualitatively new behaviors (e.g., multi-step arithmetic, code synthesis, or chain-of-thought reasoning) that appear only once models cross certain size or data thresholds [4–6]. These behaviors are often brittle: performance can vary dramatically under small prompt perturbations, domain shifts, or adversarial red-teaming. Systematic benchmarks such as BIG-bench and HELM have shown that different models occupy quite different "capability profiles" across tasks and metrics, even when they are similar in size and training tokens [6,10]. In other words, parameter count and dataset size are poor *coordinates* for reasoning about what a model can actually do.

This mismatch has significant consequences. In practice, both research and deployment decisions are often guided by aggregate metrics (e.g., average accuracy across tasks or overall perplexity) and informal narratives of "bigger is better". However, downstream users care about *specific* capabilities—such as step-by-step scientific reasoning, safe tool use, or robust handling of low-resource languages. Foundation-model surveys and critiques have repeatedly emphasized that current training and evaluation practices make it difficult to characterize these capabilities and their failure modes in a fine-grained way [8–11]. Moreover, as models are deployed in safety-critical settings, we increasingly need guarantees that certain capabilities are *present* (for reliability) and that others are *absent or constrained* (for safety). Aggregate next-token loss offers little guidance on either.

At the same time, a growing body of work suggests that model behavior is more structured than the token-centric view implies. Studies of "task vectors" show that fine-tuning on a specific task induces a low-dimensional direction in weight space that can be added, subtracted, or composed to selectively gain or forget capabilities [12]. Benchmarks like BIG-bench explicitly organize evaluation around a diverse collection of tasks contributed by domain experts, rather than generic text prediction [6]. And emerging multi-agent and tool-augmented systems implicitly decompose complex user goals into sub-tasks handled by different agents or tools. However, these ideas are usually treated as *downstream* engineering tricks on top of a monolithic pre-training recipe, rather than as first-class objects in the design of training data itself.

The gap between token-level training and task-level capabilities becomes even more pronounced in scientific and engineering applications. Consider, for example, interfacial fluid dynamics and wetting phenomena. Recent studies have systematically explored how surface anisotropy, corrugation, and sharp edges affect static contact angles and droplet morphologies [20–22], and how interfacial reactions on liquid metal substrates can drive spontaneous droplet motion, rotation, and multi-lobed dynamics [23–25]. Other work has quantified the influence of substrate temperature, vapor-sink effects, and micro/nano-structured surfaces on dry-zone formation, condensation patterns, and anti-icing performance [26–28]. These problems demand coherent reasoning across continuum mechanics, thermodynamics, numerical simulation, and experimental design. A scientific LLM assistant that merely "saw" many tokens from these articles would not automatically acquire the *task-specific* capabilities needed to, say, derive governing equations, choose appropriate dimensionless groups, or design new droplet-rotation experiments. Capability must be cultivated at the level of tasks, not just exposure to related text.

This observation generalizes far beyond fluid mechanics. Across domains such as materials discovery, molecular biology, and control systems, scientific workflows can be decomposed into structured task graphs—hypothesis generation, experimental planning, code and simulation setup, data interpretation, and iterative refinement. The current pre-training paradigm treats all textual traces of these workflows as an unlabelled mixture, hoping that a sufficiently large model will implicitly disentangle and master each step. This "capability-oblivious" approach is increasingly showing diminishing returns: ever-larger models show only modest improvements on many scientific benchmarks, while significant blind spots and failure modes remain [6,9,10].

 doi:10.20944/preprints202512.0499.v1

In this Perspective, we argue that the field needs to re-center training data and objectives around *task and capability space*, rather than only tokens and parameters. Concretely, we propose to represent the space of desired behaviors as a *capability graph*: a structured set of tasks and sub-tasks, with edges encoding relationships such as pre-requisites, compositionality, and potential negative transfer. Training data, synthetic data generation, and optimization objectives should then be organized around this graph. Instead of asking "how many tokens do we have?" or "what is the compute budget?", we should ask "which regions of capability space are covered with sufficient diversity and difficulty?", "which capabilities are over- or under-represented?", and "how should we allocate marginal data and compute to expand the frontier of useful abilities?".

A capability-centric view has several potential advantages. First, it provides a natural language for designing *curricula* over tasks—progressions from simpler to more complex capabilities, with explicit attention to transfer and interference. Second, it allows us to leverage emerging tools such as task vectors and multi-agent architectures as operating on explicit subgraphs of capability, rather than opaque model deltas. Third, it provides a more faithful interface to evaluation and safety: benchmarks like HELM and BIG-bench can be mapped onto the capability graph, and red-teaming can be framed as probing specific regions (e.g., deception, bio-risk, or model self-modification) [6,10,11]. Finally, for scientific applications, a capability graph aligns naturally with how human researchers think about expertise—mechanics vs. thermodynamics, experiment vs. simulation—and can guide the construction of living, domain-specific training ecosystems.

The remainder of this Perspective develops this argument in four steps. Section 2 analyzes the misalignment between token/parameter-centric scaling laws and the actual capability profiles that matter in practice. Section 3 discusses how to construct capability graphs from real-world task taxonomies, benchmarks, and interaction logs. Section 4 explores curriculum and data-generation strategies over capability space, including how to use synthetic data to selectively strengthen or attenuate abilities. Section 5 examines how multi-agent systems and task vectors can be interpreted as specialization on subgraphs of capability, rather than ad-hoc modularization. Section 6 sketches how evaluation, red-teaming, and safety guarantees might be reformulated in capability-centric terms, and outlines open research questions for building such systems in scientific domains such as the droplet and interfacial-flow problems mentioned above.

2. Why Tokens and Parameters Misalign with Capabilities

The current LLM paradigm treats language modeling as a large-scale curve-fitting problem over internet-scale text. Models are specified by a small set of coarse-grained knobs—number of parameters, number of training tokens, optimizer and learning-rate schedule—and progress is measured primarily by aggregate loss curves and a handful of headline benchmarks [1–3]. This abstraction has been remarkably effective for engineering, but it creates a conceptual gap: none of these quantities directly describe *which capabilities* are being learned, or how they interact. In this section, we unpack several reasons why tokens and parameters are only weakly aligned with the structure of capability space.

First, training data are not a neutral "sea of tokens", but an extremely skewed mixture of tasks, domains, and interaction styles. Web-scale corpora massively over-represent casual conversation, duplicated boilerplate, and templated content, while under-representing complex scientific reasoning, multi-step planning, or rare but safety-critical behaviors [8,9]. Because standard pre-training minimizes a uniform next-token loss over this mixture, the optimization process implicitly favors capabilities that improve loss on *frequent and easy* patterns, even if those are not the capabilities we care most about downstream. As a result, two models trained on the same number of tokens can have very different capability profiles depending on the composition and curation of those tokens [9–11].

Second, scaling laws themselves reveal that token count and parameter count are only *average-case* predictors. Kaplan-style scaling shows smooth power-law improvements of cross-entropy loss as we grow model and data size [2], and compute-optimal analyses refine this picture [3], but these

 doi:10.20944/preprints202512.0499.v1

curves aggregate over all prediction events. When we look at specific capabilities—multi-step arithmetic, compositional generalization, or tool use—performance often improves in discrete, step-like transitions, or remains flat until a threshold is crossed [4,5,6,11]. The "emergent abilities" literature makes this point explicitly: capabilities appear or disappear abruptly as we vary model size or training mix, even though aggregate loss changes smoothly [4]. In other words, capability space has *phase transitions* that are invisible at the level of tokens and parameters.

Third, token-centric training encourages shortcut solutions and spurious correlations that undermine robust capability. Because the objective is to minimize next-token prediction error, models can exploit dataset artifacts that correlate with labels or answers without acquiring the underlying reasoning skill. This has been documented in many benchmarks where LLMs appear strong under one evaluation protocol but fail under adversarial perturbations, counterfactuals, or distribution shifts [8,10,11]. For example, chain-of-thought prompting can elicit seemingly sophisticated reasoning, but subsequent work shows that models often memorize surface patterns of explanations rather than building stable internal procedures [5,10,11]. From a capability perspective, the model has learned a "style of explanation" capability without the corresponding "algorithmic reasoning" capability—something that token-level metrics cannot distinguish.

Fourth, the mapping from tokens to capabilities is many-to-many and highly context-dependent. A single capability (e.g., "write correct Python functions that interact with files and APIs") is supported by a wide variety of textual evidence scattered across documentation, tutorials, Q&A threads, and code repositories. Conversely, the same textual pattern can support multiple capabilities depending on how it is used in a task. Standard pre-training ignores this structure: all occurrences of a token sequence contribute equally to the loss, regardless of whether they are part of a genuine problem-solving trajectory, copied boilerplate, or a degenerate case. Fine-tuning and instruction-tuning try to correct this by reweighting specific examples [7,9,11], but they operate on tiny slices of data compared to the pre-training corpus, and they still lack an explicit representation of which capability each example is supposed to train.

Fifth, parameter count is an even poorer proxy for capability than token count. While larger models have greater representational capacity and thus can, in principle, implement more complex behaviors, the *allocation* of this capacity across capabilities is shaped by training signals, not by parameters alone. Two models with the same size but different data distributions or training curricula can exhibit drastically different strengths and weaknesses [6,9,10]. Moreover, recent work on parameter-efficient fine-tuning, task vectors, and model editing shows that relatively small parameter deltas can substantially alter a model's behavior on specific tasks [12,15–17]. This suggests that "capability subspaces" are localized and structured in weight space, and that global quantities like parameter count tell us little about which of these subspaces are occupied.

Sixth, the temporal dynamics of training do not align cleanly with the acquisition of capabilities. During pre-training, models encounter tasks in an effectively random interleaving. There is no explicit notion of pre-requisite skills, curriculum, or consolidation. Yet, many capabilities—such as multi-step mathematical reasoning, code synthesis, or scientific planning—are compositional and hierarchical, with natural pre-requisite structures. In scientific domains, for example, correctly interpreting droplet dynamics in wettability experiments requires prior competence in fluid mechanics, non-dimensional analysis, and experimental uncertainty; planning a droplet-rotation experiment on liquid metal surfaces further demands familiarity with interfacial chemistry and reactive wetting [20–25]. The existing token-centric pipeline assumes that random exposure to all these ingredients will eventually yield the composite capability, but this is a strong and empirically questionable assumption, especially when certain sub-tasks are sparsely represented or confounded.

Seventh, token-centric pre-training interacts in complex ways with *post-training* objectives such as RLHF and safety fine-tuning. These stages introduce new, often misaligned pressures: RLHF tends to reward helpfulness and harmlessness in conversational settings [7,11], while domain-specific fine-tuning pushes towards accuracy and coverage on specialized tasks. Without an explicit representation of capabilities, these signals can interfere in unpredictable ways. For instance,

 doi:10.20944/preprints202512.0499.v1

alignment training may attenuate useful but "risky-looking" capabilities (e.g., critiquing code or experimental designs too aggressively), while amplifying superficial politeness. Conversely, task-specific fine-tuning can revive or amplify unsafe behaviors learned in pre-training if they are entangled with the target domain [8,9]. From a system-level perspective, we are editing a high-dimensional capability manifold through local gradient updates, with little visibility into which regions are being strengthened or weakened.

Eighth, evaluation regimes today mostly inherit the same misalignment. Benchmarks are typically reported as aggregate scores—averages over diverse tasks, domains, and metrics—which mask the uneven landscape of capabilities [6,10,13,14]. HELM, BIG-bench, and similar efforts make this problem explicit by decomposing performance into slices across tasks and risk dimensions [10,13,14], but even here, we lack a *structural* representation of how tasks relate to one another, which capabilities they exercise, and where transfer should occur. In scientific applications, this leads to gaps where a model appears competent on benchmarked tasks (e.g., summarizing fluid mechanics papers or answering basic wetting questions), yet fails dramatically when asked to design a new condensation experiment or reason about multi-physics couplings such as Kelvin–Raoult–Stefan effects in confined geometries [26–28].

Finally, there is a practical cost: capability-oblivious scaling leads to inefficient use of data and compute. If we do not know which capabilities are underdeveloped, we cannot target them with additional data collection, synthetic data generation, or specialized training. We instead default to uniform expansion—more tokens of everything, bigger models for everything. This is untenable in domains where high-quality data are scarce or expensive (e.g., proprietary industrial logs, lab experiments, or specialized simulations), and it also raises safety concerns, since indiscriminate scaling may strengthen unwanted capabilities alongside desired ones [8,9,18].

Taken together, these observations motivate a transition from token- and parameter-centric thinking to a capability-centric framework. The point is not that tokens and parameters are irrelevant—they remain essential for engineering and scaling—but that they should be treated as *resources* deployed strategically across a structured capability space, rather than as the primary coordinates in which we describe progress. In the next section, we introduce capability graphs as a concrete way to represent this space: nodes representing task families or skills, edges encoding pre-requisites and transfer, and annotations capturing risk, data availability, and desired robustness. Such graphs provide the missing link between low-level training signals and high-level behaviors, enabling more deliberate control over what our models actually learn.

3. Representing Task Space as Capability Graphs

If tokens and parameters are poor coordinates for understanding what an LLM can do, we need a richer representation of *task and capability space*. In this section, we propose capability graphs as such a representation. Intuitively, a capability graph treats tasks and skills as nodes, and encodes their relationships—pre-requisites, composition, transfer, and interference—as edges. This structure provides a bridge between low-level training signals and high-level behaviors, enabling more deliberate control over how models acquire, maintain, and combine capabilities.

We start by distinguishing *tasks* from *capabilities*. A task is an externally specified input–output mapping under a given protocol—e.g., "solve GSM8K-style math word problems," "write a Python function with tests," or "design an experiment to measure droplet rotation on a liquid-metal surface given certain constraints." Capabilities are the internal skills that support performance across many tasks—such as symbolic manipulation, causal reasoning, tool orchestration, or pattern recognition in interfacial-flow simulations. In practice, tasks and capabilities are entangled: tasks are the observable probes through which we infer capabilities; capabilities are latent constructs that explain performance patterns across tasks. Any useful representation must therefore capture both the *granularity* (micro vs. macro skills) and the *shared structure* across tasks.

A capability graph does this by treating nodes as "task families" or "skills" at an appropriate level of abstraction and edges as relations between them. At the micro level, nodes might correspond

 doi:10.20944/preprints202512.0499.v1

to operations like "integer addition," "loop indexing in Python," or "extract relevant variables from a natural-language problem description." At the meso level, nodes might be families such as "multi-step arithmetic reasoning," "file I/O and API usage in code," or "dimensional analysis in fluid mechanics." At the macro level, nodes can represent complex workflows like "conduct a parameter-sweep study of droplet contact-line pinning," which decomposes into literature search, hypothesis formulation, simulation setup, experiment planning, and data interpretation [20–28]. Edges express how these nodes depend on and interact with each other—for example, "multi-step arithmetic" requires "integer addition" and "subtraction," or "experiment planning for droplet rotation" requires "dimensional analysis" plus "basic interfacial chemistry".

Constructing such a graph can be approached in several complementary ways. One route is *top-down*, starting from human-designed taxonomies. Work on curriculum learning explicitly organizes examples from simple to complex based on human notions of difficulty, implicitly assuming a partial order over tasks [31]. Similarly, the multi-task learning literature treats "tasks" as related but distinct prediction problems that share internal representations [32]. In scientific domains, an instructor or domain expert could outline a prerequisite graph over capabilities: for instance, in interfacial fluid mechanics, basic Navier–Stokes reasoning, Young's equation and wetting, Kelvin and Raoult effects, and then coupled Kelvin–Raoult–Stefan transport in confined geometries [20–22,26–28]. This yields an initial capability graph that can guide data collection and evaluation even before any model is trained.

A second route is *bottom-up*, inferring structure from empirical transfer patterns. In computer vision, Taskonomy demonstrated that we can model the space of tasks by measuring how well representations learned on one task transfer to others, then using these transfer scores to build a directed task graph [33]. Related approaches in multi-task and modular reinforcement learning use "policy sketches" or shared modules across tasks to discover subtask structure [34]. For LLMs, an analogous procedure would treat each evaluation task (e.g., MATH, GSM8K, coding benchmarks, scientific question answering, experiment design) as a node, then estimate directed edges based on fine-tuning transfer: if fine-tuning on task A yields large gains on task B but not vice versa, that suggests a pre-requisite relation A → B. By sampling over many tasks and measuring asymmetric transfer, we can construct a data-driven capability graph that reflects how skills cluster and compose in the current model.

A third route is *interaction- and log-based*. Deployed models generate rich traces of interactions: multi-turn conversations, tool calls, external code execution, and user feedback. These traces naturally decompose into sub-tasks, often with explicit structure: the model first reformulates the problem, then queries a database, then analyzes results, then proposes follow-up actions. With suitable instrumentation, we can segment these traces into labeled "sub-task spans" and cluster them into recurring capability patterns—e.g., "planning," "retrieval orchestration," "simulation setup," "error diagnosis"—using representation learning or sequence clustering. Over time, transitions between these sub-task clusters define a Markov-like structure over capabilities, which can be summarized as a directed graph. Failure modes (e.g., hallucinated citations, unstable experimental suggestions) then appear as particular nodes or edges with high error rates, highlighting parts of the graph that require targeted training or safety interventions [6,9–11].

These three routes are complementary rather than exclusive. Human-designed curricula inject domain knowledge and interpretability; transfer-based graphs capture emergent structure that humans might miss; interaction-based graphs incorporate real deployment conditions and user priorities. An attractive strategy is to start from a coarse human-defined skeleton and refine it with empirical evidence: merging nodes that behave identically under transfer, splitting nodes that show heterogeneous error patterns, and adding edges where transfer is unexpectedly strong or weak. In scientific settings, for example, we might initially treat "droplet impact dynamics analysis" as a single node, but later split it into "regime classification", "scaling-law derivation", and "COMSOL or CFD setup" once interaction logs reveal distinct failure signatures across these sub-capabilities.

 doi:10.20944/preprints202512.0499.v1

A key design choice is *granularity*: how fine-grained should the nodes be? Too coarse, and the graph collapses into a handful of generic skills ("reasoning", "coding") that are too broad to be actionable. Too fine, and the graph becomes intractable to curate or use, with thousands of atomic skills and sparse data per node. Here, insights from curriculum learning and multi-task learning are useful. Curriculum learning suggests that capabilities should be defined at levels that support meaningful progression in difficulty and abstraction [31]. Multi-task learning shows that tasks which share internal structure (e.g., similar input modalities and label semantics) benefit from being grouped under shared representations [32]. Combining these, we can define capability nodes as clusters of tasks that (i) exhibit strong positive transfer, (ii) occupy a similar "difficulty band" along likely learning trajectories, and (iii) are meaningful units for downstream stakeholders (e.g., "design valid lab protocols" vs. "output syntactically correct LaTeX").

Beyond topology, capability graphs can be *annotated* with rich metadata. Each node can carry properties such as (i) estimated competence and robustness of the current model (e.g., accuracy with confidence intervals across evaluation suites), (ii) data coverage (how many and what kinds of examples exist for training and evaluation), (iii) domain and risk level (e.g., harmless chit-chat vs. biosecurity-sensitive planning), and (iv) resource cost (e.g., whether exercising the capability requires expensive simulations or lab experiments). Edges can be annotated with transfer strength (how much do we gain on B by training on A?), interference risk (does training on A degrade performance on C?), and compositional relations (which nodes must be activated jointly to solve a particular macro-task?). These annotations turn the graph into a quantitative object that can drive optimization and governance, not just a conceptual diagram.

For scientific LLMs, capability graphs can also be *multi-layered*, reflecting the interplay between general linguistic skills and domain-specific expertise. A high-level layer might include generic capabilities like "mathematical proof sketching", "data table interpretation", or "tool-augmented literature search". Beneath each of these, domain-specific subgraphs capture specializations: within "mathematical reasoning", branches for asymptotic analysis, scaling laws, and stability analysis; within "data interpretation", branches for analyzing droplet condensation images, interpreting interfacial shear stress distributions, or reading spectroscopy data [20–28]. This layered structure allows us to reason about transfer across domains (e.g., from fluid mechanics to materials science) while still respecting the specificity of scientific practice.

Importantly, capability graphs are not static. As models, data, and usage patterns evolve, the graph should be updated. New tasks appear (e.g., interacting with novel tools or experimental platforms), which may require adding nodes and edges. Some capabilities that seemed distinct may turn out to be tightly coupled and better represented as a single node; others may need to be split as we discover distinct failure modes. This continual refinement parallels how human disciplines evolve their own taxonomies (e.g., the progressive differentiation of "fluid mechanics" into subfields like microfluidics, non-Newtonian flows, and interfacial phenomena), and it aligns with the broader shift from static corpora to living data ecosystems discussed elsewhere in this series.

From the perspective of training and deployment, the key advantage of a capability graph is that it provides *handles* for targeted intervention. Instead of "add 100B more tokens," we can say "node X (e.g., multi-step experimental planning in wetting problems) exhibits low robustness and sparse data; we should collect more diverse examples, generate synthetic variants, and adjust training to improve competence along this node and its immediate neighbors." Instead of a monolithic safety fine-tuning stage, we can implement *capability-aware* guardrails: constraining or disabling particular nodes (e.g., high-risk bioexperimental planning) while allowing others to remain fully expressive, and monitoring edge activations when hazardous combinations of capabilities are invoked [8–11,18]. In short, the capability graph is the structural prior we were missing: a map of what we want models to be able to do and how those abilities relate.

The next section will build on this representation to discuss *capability-centric training*: how curricula, data selection, and synthetic data generation can be organized over the capability graph,

 doi:10.20944/preprints202512.0499.v1

and how multi-objective loss functions can explicitly trade off breadth, depth, robustness, and safety at the level of capabilities rather than raw tokens.

4. Capability-Centric Training: Curricula, Data, and Loss Functions

Once we have a capability graph, the natural next step is to *train with respect to it*. Instead of treating training as minimizing a single scalar loss over a homogeneous token stream, we can view it as shaping performance over a structured set of capability nodes and edges. In this section, we outline three pillars of capability-centric training: (i) curricula over the graph, (ii) data selection and synthetic data guided by capability gaps, and (iii) multi-objective loss functions defined at the level of capabilities rather than raw tokens.

Curricula over capability space. Curriculum learning has long argued that models learn better when exposed to examples in an organized progression from simple to complex [31,35,36]. In the capability-graph view, such progressions become explicit *paths* through the graph. Early training focuses on foundational nodes (e.g., lexical competence, basic syntax, simple arithmetic), then gradually activates higher-level nodes that depend on them (e.g., multi-step reasoning, code synthesis, experimental design). Edges encode pre-requisites and transfer; the curriculum is essentially a schedule that decides when to introduce or revisit each node. Unlike traditional curricula defined in terms of example difficulty alone [31], a capability-centric curriculum can consider how improvements on one node are expected to propagate to its neighbors, and it can actively avoid sequences that create negative transfer.

Practically, this suggests splitting training into phases that emphasize different regions of capability space. A base pre-training phase may still operate on broad coverage of the graph, but with explicit guarantees that certain foundational nodes achieve a minimum level of robustness before "unlocking" more demanding capabilities. Later phases can be more surgical, prioritizing nodes with strong application value (e.g., scientific planning, tool-augmented coding) or high safety salience (e.g., refusal to perform risky experiments) [8–11,18]. In scientific domains, a curriculum might first ensure competence on mechanics, thermodynamics, and basic wetting theory, then gradually introduce more specialized capabilities such as Kelvin–Raoult–Stefan coupling in confined geometries or planning droplet-rotation experiments on liquid metal surfaces [20–28]. Instead of hoping these composite skills emerge implicitly from exposure, the curriculum treats them as explicit milestones.

Curriculum design can itself be *adaptive*. Work on automated curricula and reinforcement-learning curricula shows that it is possible to select training tasks online based on the agent's learning progress, sampling more heavily from tasks where error is high but improving, and less from those where learning has plateaued [35,36]. In a capability graph, this corresponds to sampling nodes and edges based on local learning signals: if performance on a capability node is stagnant despite ample data, training might shift to its ancestors (to reinforce prerequisites) or to closely related siblings (to exploit transfer). Conversely, if a capability is mastered but is a prerequisite for many high-value downstream nodes, we may still allocate occasional training to prevent forgetting and maintain the "bridge" across the graph.

Data selection and active filling of capability gaps. A capability graph also changes how we think about data. Rather than asking "how many tokens do we have?", we can ask "how many *useful* examples do we have for each capability node, and how diverse are they?". Recent work on data pruning shows that carefully selecting a subset of training examples can yield better performance than naive scaling, effectively beating naïve neural scaling laws [37]. Active learning and data valuation similarly emphasize that not all examples are equally informative [37,39]. In a capability-centric regime, these ideas become capability-aware: data are tagged (explicitly or implicitly) with the capabilities they exercise, and selection policies aim for balanced, diverse coverage of the graph under a compute budget.

Concretely, one can maintain per-node statistics: approximate sample counts, diversity metrics (e.g., lexical or structural variety), and sensitivity measures (how much does training on this node

 doi:10.20944/preprints202512.0499.v1

change downstream performance?). Nodes with sparse or low-quality data are prime candidates for targeted collection or synthetic augmentation; nodes with redundant or low-impact data can be down-weighted or pruned [37]. This is especially important for scientific domains where high-quality labeled data (e.g., carefully annotated wetting experiments, validated COMSOL simulations, or lab protocols) are scarce and expensive. The capability graph makes explicit which scientific capabilities are under-served by current corpora—for instance, "deriving scaling laws for droplet impact from raw experimental plots" may have far fewer examples than "summarizing review articles on wettability"—and helps prioritize data collection accordingly.

Synthetic data and self-supervision over the graph. Synthetic data generation has become a central tool for aligning and extending LLMs, from instruction-following to safety training [7,29,38]. In a capability-centric setting, synthetic data generation can be driven by *capability gaps*: nodes or edges where performance is low, data are scarce, or risk is high. Self-instruct methods, where a model generates its own instructions and solutions which are then filtered and used for further training, can be adapted to operate at the level of capability nodes [38]. For a specific scientific capability—say, "design parameter-sweep experiments to probe contact-line pinning on corrugated substrates" [21,22]—we can prompt a strong teacher model (or a committee of models) to generate diverse problem instances and solutions, then vet them automatically or via human experts. These synthetic examples are then added as training signals for that node and its neighbors.

Critically, synthetic data must respect the graph's *risk annotations*. For high-risk capabilities (e.g., bio-experimental planning), synthetic data generation may aim to *sharpen refusals* and safe decompositions, rather than to expand capability. For low-risk scientific reasoning (e.g., deriving dimensionless groups for a new microfluidic configuration [28]), synthetic generation can safely explore a broad combinatorial space of problem setups, boundary conditions, and parameter ranges, vastly amplifying the effective data budget for these nodes. Synthetic tasks can also be composed along edges: to train a macro-capability like "end-to-end droplet-rotation experiment planning", we can synthesize multi-step trajectories that explicitly invoke sub-capabilities such as literature retrieval, hypothesis formation, and parameter selection [23–25].

Multi-objective, capability-aware loss functions. The capability graph also invites a shift from a single scalar loss to a *vector* of capability-specific objectives. Each node can define its own loss (e.g., task-specific cross-entropy, pass@k on code, constraint violation penalties, or safety violation rates), and training becomes a multi-objective optimization problem over this vector. Scalarization—choosing weights for each node's loss—then expresses the priorities of stakeholders: which capabilities we care most about, which safety constraints are hard vs. soft, and how we trade off breadth (coverage across many nodes) versus depth (robustness on a subset) [32–34].

In practice, we do not need a separate head or loss term for every node in a massive graph. Nodes can be grouped into clusters (e.g., "mathematical reasoning", "scientific planning", "code and tools", "conversation and instruction-following") with shared losses, while a smaller number of sentinel nodes—high-risk or high-value capabilities—are monitored individually. Techniques from multi-task learning and gradient surgery can help manage conflicting gradients between capabilities, reducing destructive interference [32–34]. For instance, if optimization steps that improve a high-risk capability also degrade a safety-critical refusal behavior, gradient alignment methods can detect and mitigate this conflict before it manifests as a regression in deployment.

A capability-aware loss also integrates naturally with RLHF and related post-training methods [7,10,11]. Instead of a single reward model that collapses "helpfulness" and "harmlessness" into one scalar, we can train multiple reward heads aligned with different regions of the capability graph (e.g., one focusing on scientific accuracy, another on stylistic alignment, another on safety constraints). The RL objective then becomes a weighted combination of these rewards, making it easier to reason about how changes in weights will affect specific capabilities. In principle, we can even treat certain capabilities as *hard constraints*—e.g., zero tolerance for unsafe experimental suggestions in bio-domains—using constrained RL or Lagrangian methods, while optimizing softer objectives like style or verbosity.

 doi:10.20944/preprints202512.0499.v1

Continual learning and preventing capability drift. Capability-centric training must also confront continual learning and forgetting. As models are updated with new data and objectives, performance can drift in ways that are hard to detect when monitoring only aggregate metrics. The capability graph provides a scaffold for continual evaluation: key nodes and edges can be periodically re-assessed on held-out benchmarks or canary tasks to detect regressions. When a regression is detected, model-editing methods [14–16] or targeted fine-tuning on affected nodes can be used to repair capabilities while minimizing collateral damage. Because capabilities are localized in the graph, interventions can be more focused than global re-training.

In scientific applications, where models may be continuously updated with new experimental data, simulation results, or lab-specific protocols, such continual capability tracking is essential. For example, as new studies on interfacial reactions and droplet dynamics are incorporated [20–28], we want to ensure that the model's existing competence on core fluid-mechanics reasoning is preserved, while its specialized capabilities (e.g., planning new self-rotating droplet experiments or interpreting fractal microchannel flows) improve. Capability-aware curricula, data selection, and loss functions offer a principled route to achieve this, turning the capability graph from a static taxonomy into a living control surface for training.

In summary, capability-centric training reinterprets data and optimization as tools for sculpting performance over a structured capability space. Curricula become paths through the graph; data selection and synthetic generation become operations that thicken or extend specific regions; loss functions become multi-objective, reflecting stakeholder priorities and safety constraints. The next section builds on this view to discuss system-level architectures—particularly multi-agent systems—in which different models or modules specialize on subgraphs of capability and are orchestrated by a higher-level planner.

5. System-Level Architectures: Multi-Agent Specialization on Capability Subgraphs

Up to this point, we have treated the capability graph largely as a way to reason about *one* model's skills. In practice, however, most high-value LLM applications are already *systems*: a backbone model surrounded by tools, retrieval components, code interpreters, simulators, or even other LLMs. Multi-agent frameworks such as AutoGen explicitly embrace this systems view, letting developers compose multiple agents that converse and collaborate to solve tasks [41]. The capability-graph perspective provides a natural way to structure such systems: different agents specialize on *subgraphs* of capability and coordinate through well-defined interfaces, rather than implicitly entangling all behaviors inside a single monolithic model.

There are at least three reasons to embrace multi-agent specialization. First, it aligns with how human expertise is organized: we do not expect a single scientist to be simultaneously world-class in PDE analysis, microfluidic chip design, and regulatory review; instead, teams form around complementary skills. Second, it matches the modularity of tools and external systems. Architectures like MRKL (Modular Reasoning, Knowledge, and Language) already route queries between an LLM, calculators, search APIs, and symbolic reasoners using a router policy [42]. Third, multi-agent systems can be easier to debug and govern: if a failure can be localized to a specific specialized agent or subgraph, we can intervene surgically—re-training, constraining, or replacing that component—without destabilizing the entire system. From a capability-graph viewpoint, each agent is associated with a subset of nodes (capabilities) and edges (compositions) on which it is expected to be competent. For example, in a scientific setting we might define:

- A literature & retrieval agent specialized on "information seeking" and "evidence synthesis" nodes;
- A theory & reasoning agent covering "symbolic manipulation", "dimensional analysis", and "scaling-law derivation" capabilities;
- A simulation & coding agent focusing on "numerical setup", "code generation", and "debugging" subgraphs;

 doi:10.20944/preprints202512.0499.v1

- An experiment-planning agent that composes these capabilities into concrete protocols for droplet impact, wetting, or condensation experiments [20–28];
- A safety & governance agent that monitors high-risk capabilities and enforces domain-specific constraints.

Each of these agents might share a common pretrained backbone but be fine-tuned and evaluated on different parts of the capability graph using the training machinery of Section 4.

Architectural patterns. Existing LLM systems already implement several patterns that can be reinterpreted as specialization on capability subgraphs. MRKL-style systems use a *router* to dispatch sub-tasks to appropriate modules (e.g., calculator, search, code execution), with the LLM acting as both interpreter and glue [42]. ReAct combines internal reasoning traces with explicit actions (API calls, environment interactions), effectively splitting "think" and "act" capabilities and interleaving them over time [44]. Toolformer shows that a model can teach itself when and how to call specific tools, creating small, tool-centric capability subgraphs linked into the main language model [43]. Multi-agent frameworks like AutoGen generalize these ideas: agents can themselves wrap LLMs, tools, or humans, and a conversation protocol coordinates them to solve tasks that are too complex for a single agent [41,46]. Within the capability-graph lens, these patterns correspond to different ways of *partitioning and composing* subgraphs:

1. Router + specialists (MRKL style). The capability graph is partitioned into regions handled by different specialists (calculator for arithmetic, retriever for factual recall, LLM for open-ended reasoning). A router agent learns a mapping from user queries (and intermediate states) onto these regions.
2. Planner + workers (AutoGen / task-decomposition style). A planner agent operates at a higher level of the capability graph, decomposing a macro-task into a sequence of sub-tasks (nodes/edges). Worker agents specialize on narrower subgraphs (e.g., "code and run simulations", "summarize experimental literature"), with the planner deciding which worker to invoke when.
3. Hierarchical controllers. Multiple layers of agents correspond to different abstraction layers of the capability graph: top-level agents reason about high-level strategy (which capabilities to invoke in what order), mid-level agents handle specific domains (fluid mechanics vs. materials vs. data analysis), and low-level agents interface with concrete tools (CFD solvers, lab robots, databases).

Training specialization on subgraphs. Capability-centric training provides a recipe for *how* to specialize these agents. Instead of training all agents on the same mixed corpus, we assign each agent a tailored curriculum and data distribution focused on its subgraph. A simulation agent, for instance, would be trained more heavily on code repositories, COMSOL/CFD templates, and problem–solution pairs about grid independence or stability, while the theory agent would see more mathematical derivations and scaling-law analyses [20–22,26–28]. Evaluation is likewise subgraph-specific: the simulation agent is tested on pass@k for code problems and consistency of simulation setups; the experiment-planning agent is evaluated on protocol viability and adherence to safety constraints. Crucially, specialization does not imply isolation. Agents still need to *compose* capabilities across subgraphs. This is where inter-agent protocols—conversation, tool calls, shared scratchpads—become important. ReAct-style traces, where an agent alternates between reasoning and acting, can naturally extend to multi-agent settings: a planner agent reasons about next steps, delegates an action to a specialist agent (e.g., "run a 2D axisymmetric droplet impact simulation at Weber number 100"), then incorporates the returned observations into subsequent reasoning [44]. The capability graph tells us which compositions are legitimate or desirable: edges between nodes assigned to different agents correspond to interfaces that must be supported by the interaction protocol.

 doi:10.20944/preprints202512.0499.v1

Tool use as micro-agents. In many systems, tools are treated as passive APIs, but from a capability standpoint they are *micro-agents* with very narrow, high-precision capabilities (e.g., "evaluate an integral", "query a database", "control a lab instrument"). Toolformer demonstrates that LLMs can learn to decide when and how to call such tools in a self-supervised way [43]. The capability graph can treat each tool as a small node or cluster of nodes—"calculator capability", "CFD solver invocation capability"—and track how often they are used, how reliable their outputs are, and how they interact with higher-level reasoning capabilities. In scientific workflows, this includes not only numerical tools but also physical hardware: robotic pipetting stations, thermal chambers, droplet imaging systems, etc. A multi-agent lab assistant could route "wet" tasks (executing condensation experiments, adjusting substrate temperature [26,27]) to embodied agents while keeping high-level reasoning in purely virtual agents. Benefits: modularity, interpretability, and safety. Specialization on capability subgraphs has several system-level advantages.

- Modularity and maintainability. Because each agent is responsible for a subset of capabilities, we can update or replace it independently. If a new, better simulation engine is available, only the simulation agent and its graph region need to be retrained and re-certified.
- Interpretability. When a failure occurs—e.g., an incorrect scaling law for droplet spreading or an unsafe experimental suggestion—it can often be attributed to a specific agent/subgraph: the theory agent's reasoning, the planner's decomposition, or the safety agent's oversight. This makes post-mortems and fixes more targeted.
- Safety and capability isolation. High-risk capabilities (bio-lab planning, chemical synthesis, security-relevant code) can be isolated into specialized agents wrapped in strong guardrails and access controls. Other agents can be prohibited from calling them directly, forcing requests through an oversight layer. MRKL-style routers and AutoGen-style orchestrators can implement policy checks at routing time [41,42,45].

Risks and open problems. Multi-agent systems also introduce new risks. Coordination failures can produce emergent misbehavior even if each individual agent is reasonable; planning agents may over-delegate or under-specify tasks, leading to error cascades. Capability subgraphs that look benign in isolation may become hazardous when combined—for instance, coupling strong experimental-design capabilities with tool agents that control physical hardware. Auto-generated agents (e.g., using code-generation to create new worker agents on the fly) raise additional questions about vetting and bounding their capabilities [41,46].

The capability graph helps here by making these compositions explicit. Inter-agent protocols correspond to traversals that cross subgraph boundaries; we can require that certain high-risk traversals either be forbidden (no edge in the graph) or mediated by a safety node (an oversight agent). Evaluation and red-teaming can then systematically target not only individual agents but also *interfaces* between them, probing, for example, whether a planner can trick a safety agent into approving disallowed experiments or whether a simulation agent can be induced to mask uncertainty.

For scientific applications—such as automated exploration of wetting, condensation, and droplet-impact phenomena—the endgame is not a single "god model" but a *colony* of specialized agents: some deeply versed in theory and simulation, others in experimental execution and control, and still others in safety and governance. The capability-graph framework offers a blueprint for designing, training, and coordinating such colonies: it tells us how to carve up the space of capabilities, where to place interfaces, and how to monitor and adapt the system as new domains and tools come online. In the next section, we will see how the same framework can reshape evaluation, red-teaming, and safety guarantees, moving from aggregate scores to structured coverage and risk maps over capability space.

 doi:10.20944/preprints202512.0499.v1

6. Redefining Evaluation, Red-Teaming, and Safety in Capability Terms

If training is reorganized around capability graphs, evaluation and safety cannot remain token-centric either. Today, most reports still emphasize single-number metrics—average accuracy across benchmarks, win-rates in head-to-head comparisons, or coarse "GPT-4–level" labels [1,6,11]. These aggregate views obscure where a model is strong, where it is brittle, and where it poses risk. A capability-centric perspective suggests a different goal: *mapping* and *monitoring* performance over capability space, with explicit attention to high-risk regions and their interfaces. Current evaluation frameworks already hint at this direction. HELM, BIG-bench, and related efforts decompose performance across tasks, domains, and risk categories, emphasizing multi-dimensional evaluation and documenting trade-offs [6,10,13,14]. Foundation-model reports similarly catalog diverse behaviors and social impacts, arguing that narrow benchmarks are inadequate for systems with broad capabilities [8,9,18]. However, these efforts still lack an explicit structural prior: tasks are listed and grouped, but the *relations* between them—pre-requisites, transfer paths, dangerous compositions—remain implicit. As a result, coverage is uneven, blind spots are hard to predict, and safety guarantees are difficult to phrase beyond "we tried these tests and observed no catastrophic failures".

A capability graph allows us to define evaluation in terms of *coverage* and *robustness* over nodes and edges. For each capability node, we can maintain: (i) a *competence* estimate (accuracy, success rate, or calibration on relevant benchmarks); (ii) a *robustness* estimate (sensitivity to perturbations, adversarial prompts, or domain shift); and (iii) a *risk profile* (whether exercising this capability can directly cause harm, or can amplify harm when combined with others). Edges—compositions of capabilities—can be assessed for *compositional robustness*: does good performance on A and B imply good performance on "A then B", or do new failure modes appear when they are combined?

Evaluation thus becomes a coverage problem: given a capability graph and a budget of human and computational resources, how do we design tests that (a) cover high-value, high-risk nodes and edges, (b) probe likely failure regions suggested by transfer structure, and (c) provide early warning as models or training regimes change? This is analogous to test-suite design in software engineering, but with an explicit capability topology rather than a flat list of functions. For scientific LLMs, nodes might include "derive scaling laws for droplet impact from physical assumptions", "design a stable COMSOL model for a given wetting geometry", or "plan an experiment to measure condensation-induced dry zones on structured surfaces" [20–22,26–28]. Each node can be tested with a battery of domain-specific tasks; their compositions (e.g., "design + simulate + critique" loops) become edges that warrant their own tests.

Red-teaming fits naturally into this picture as *adversarial exploration of high-risk subgraphs*. Recent work on systematic red-teaming of LLMs shows that many harmful behaviors only appear under adversarial prompting and that scaling models tends to increase both capability and the surface area for misuse [8,9,47,48]. Rather than red-teaming "the model" in an undifferentiated way, a capability graph lets us target campaigns at specific regions—say, "biological experimentation planning", "chemical synthesis optimization", or "security-relevant code generation"—and at interfaces between them and benign capabilities (e.g., scientific explanation). Attackers often exploit such interfaces: they wrap harmful requests in seemingly benign tasks ("just check if this protocol is safe"), or they chain capabilities in ways that bypass local guardrails. By modeling these composites as edges, we can explicitly probe and harden them.

Safety training methods such as RLHF and Constitutional AI can also be reinterpreted through capability graphs. In the standard setup, a reward model is trained to score responses along a small set of axes (helpfulness, harmlessness, honesty), and the base model is fine-tuned to optimize this scalar reward [7,11,47]. In a capability-centric framework, rewards can be decomposed along capability dimensions: one reward head for scientific correctness on certain nodes, another for stylistic alignment, and others for different classes of safety constraints (e.g., preventing actionable wet-lab protocols vs. discouraging misleading but harmless stylistic choices). Instead of a single, entangled reward landscape, we obtain a vector of objectives tied to specific nodes and edges. This

 doi:10.20944/preprints202512.0499.v1

makes it easier to express policies such as "aggressively improve correctness on interfacial-flow reasoning [20–28] while preserving strict refusals on lab automation control" or "allow detailed discussion of Kelvin–Raoult–Stefan models but refuse any attempt to translate that into high-risk experimental procedures".

Crucially, capability graphs help articulate *negative* safety guarantees: not only what the model can do, but what it *cannot* or *must not* do. In traditional evaluation, absence of evidence (e.g., "we did not see harmful behavior in our tests") is often implicitly treated as evidence of absence, which is unsafe in adversarial settings [8,9,18]. In a capability-centric view, we instead specify that certain nodes (e.g., "high-level bio-design") are either unpopulated (not trained) or tightly wrapped by safety capabilities (e.g., strong refusal policies, supervision requirements). For those nodes, evaluation focuses less on competence and more on *leakage*: the probability that an input sequence traverses into that subgraph and produces disallowed behavior. Red-teaming then becomes a search for paths—prompt patterns, tool sequences, multi-agent dialogues—that circumvent these guardrails, which can be systematically described and monitored as graph traversals.

Multi-agent systems add another layer. As Section 5 argued, specialization on capability subgraphs can improve modularity and governance, but it also creates new interfaces and potential loopholes [41–46]. A safety agent that sits atop the graph may approve or reject actions proposed by other agents; a planner agent may obscure intents through decomposition, handing "harmless-looking" sub-tasks to worker agents that, in combination, yield a dangerous outcome. Evaluating such systems requires not only per-agent capability maps but also *joint* capability graphs for the whole system: nodes representing emergent capabilities ("end-to-end wet-lab automation", "closed-loop optimization of droplet experiments") and edges representing inter-agent protocols. Red-teaming can then target these emergent nodes directly—for instance, by asking whether the planner+simulation+experiment trio can be coaxed into suggesting experiments that exceed safe temperature limits, or that misuse condensation setups described in [23–28].

There are, of course, limitations and open challenges. First, constructing and maintaining comprehensive capability graphs is itself a hard problem (Section 3). Graphs that are too coarse will miss important distinctions; graphs that are too fine-grained will be intractable to evaluate. Second, Goodhart's law applies: once capability metrics become targets, models and training regimes will be optimized against them, potentially overfitting to benchmarks and leaving adjacent capability regions under-specified [6,10,30]. This is already visible in chain-of-thought benchmarks, where models can game the format without truly improving reasoning [5,10,30]. Capability graphs do not solve this automatically; they merely make the structure of the optimization and evaluation space explicit, which can help in designing more robust, adversarially-updated benchmarks. Third, threat models and societal norms evolve. A capability that seems benign today (e.g., detailed planning of certain types of experiments) may become problematic as new tools, lab automation platforms, or social contexts emerge. Capability graphs must therefore be *living artifacts*, updated not only with new tasks and scientific domains but also with new risk assessments [8,9,18,47,48]. Governance structures—whether institutional review boards for scientific LLMs, industry consortia, or regulators—will need to participate in defining and updating these graphs, much as they do for biosafety and data-protection standards.

Despite these challenges, capability-centric evaluation and safety offer a path beyond the current "benchmark zoo" and black-box red-teaming. For scientific LLMs, in particular, they align evaluation with how scientists actually think about expertise and risk: as structured spaces of skills that combine in lawful and sometimes dangerous ways. A capability graph that includes nodes for "asymptotic analysis of capillary waves", "planning Kelvin–Raoult–Stefan experiments in micro-gaps", and "controlling droplet-impact platforms and thermal substrates" [20–28] is not merely an abstract taxonomy; it is a map that can guide where to invest data, where to demand human oversight, and where to erect hard boundaries. Taken together, the six sections of this Perspective argue for a shift from tokens and parameters to tasks and capabilities as the primary language in which we design, train, evaluate, and govern LLMs. Tokens and parameters remain vital *resources*, but they should be

 doi:10.20944/preprints202512.0499.v1

deployed strategically over capability space, not treated as ends in themselves. Capability graphs offer a unifying scaffold: they inform curricula and data generation (Section 4), structure multi-agent architectures (Section 5), and shape evaluation and safety regimes (this section). For science and engineering applications—ranging from interfacial fluid dynamics and materials discovery to autonomous laboratories—the promise is not just more powerful models, but systems whose abilities are *legible*, *targetable*, and *governable* in terms that domain experts and society can reason about.

References

1. Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S. Language models are few-shot learners. Advances in neural information processing systems. 2020;33:1877-901.
2. Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J. and Amodei, D., 2020. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361.
3. Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D.D.L., Hendricks, L.A., Welbl, J., Clark, A. and Hennigan, T., 2022. Training compute-optimal large language models. arXiv preprint arXiv:2203.15556.
4. Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D. and Chi, E.H., 2022. Emergent abilities of large language models. arXiv preprint arXiv:2206.07682.
5. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V. and Zhou, D., 2022. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35, pp.24824-24837.
6. Srivastava, A., Rastogi, A., Rao, A., Shoeb, A.A.M., Abid, A., Fisch, A., Brown, A.R., Santoro, A., Gupta, A., Garriga-Alonso, A. and Kluska, A., 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. Transactions on machine learning research.
7. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A. and Schulman, J., 2022. Training language models to follow instructions with human feedback. Advances in neural information processing systems, 35, pp.27730-27744.
8. Bender, E.M., Gebru, T., McMillan-Major, A. and Shmitchell, S., 2021, March. On the dangers of stochastic parrots: Can language models be too big?. In Proceedings of the 2021 ACM conference on fairness, accountability, and transparency (pp. 610-623).
9. Wiggins, W.F. and Tejani, A.S., 2022. On the opportunities and risks of foundation models for natural language processing in radiology. Radiology: Artificial Intelligence, 4(4), p.e220119.
10. Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A. and Newman, B., 2022. Holistic evaluation of language models. arXiv preprint arXiv:2211.09110.
11. Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y.T., Li, Y., Lundberg, S. and Nori, H., 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. arXiv preprint arXiv:2303.12712.
12. Ilharco, G., Ribeiro, M.T., Wortsman, M., Gururangan, S., Schmidt, L., Hajishirzi, H. and Farhadi, A., 2022. Editing models with task arithmetic. arXiv preprint arXiv:2212.04089.
13. Srivastava, A., Rastogi, A., Rao, A., Shoeb, A.A.M., Abid, A., Fisch, A., Brown, A.R., Santoro, A., Gupta, A., Garriga-Alonso, A. and Kluska, A., 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. Transactions on machine learning research.
14. Meng, K., Bau, D., Andonian, A. and Belinkov, Y., 2022. Locating and editing factual associations in gpt. Advances in neural information processing systems, 35, pp.17359-17372.
15. Meng, K., Sharma, A.S., Andonian, A., Belinkov, Y. and Bau, D., 2022. Mass-editing memory in a transformer. arXiv preprint arXiv:2210.07229.
16. Tam, D., Bansal, M. and Raffel, C., 2023. Merging by matching models in task parameter subspaces. arXiv preprint arXiv:2312.04339.
17. Wiggins, W.F. and Tejani, A.S., 2022. On the opportunities and risks of foundation models for natural language processing in radiology. Radiology: Artificial Intelligence, 4(4), p.e220119.

 doi:10.20944/preprints202512.0499.v1

18. Bommasani, R., 2021. On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258.
19. Liu, H., Xue, W., Chen, Y., Chen, D., Zhao, X., Wang, K., Hou, L., Li, R. and Peng, W., 2024. A survey on hallucination in large vision-language models. arXiv preprint arXiv:2402.00253.
20. Wang, Z., Chen, E. and Zhao, Y., 2018. The effect of surface anisotropy on contact angles and the characterization of elliptical cap droplets. Science China Technological Sciences, 61(2), pp.309-316.
21. Wang, Z. and Zhao, Y.P., 2017. Wetting and electrowetting on corrugated substrates. Physics of Fluids, 29(6).
22. Wang, Z., Lin, K. and Zhao, Y.P., 2019. The effect of sharp solid edges on the droplet wettability. Journal of colloid and interface science, 552, pp.563-571.
23. Wang, Z., Wang, X., Miao, Q., Gao, F. and Zhao, Y.P., 2021. Spontaneous motion and rotation of acid droplets on the surface of a liquid metal. Langmuir, 37(14), pp.4370-4379.
24. Wang, Z., Wang, X., Miao, Q. and Zhao, Y.P., 2021. Realization of self-rotating droplets based on liquid metal. Advanced Materials Interfaces, 8(3), p.2001756.
25. Wang, Z.L. and Lin, K., 2023. The multi-lobed rotation of droplets induced by interfacial reactions. Physics of Fluids, 35(2).
26. Hu, J., Zhao, H., Xu, Z., Hong, H. and Wang, Z.L., 2024. The effect of substrate temperature on the dry zone generated by the vapor sink effect. Physics of Fluids, 36(6).
27. Hu, J. and Wang, Z.L., 2024. The effect of hygroscopic liquids on the spatial controlling of condensation on low-temperature surfaces. Surfaces and Interfaces, 55, p.105430.
28. Hu, J. and Wang, Z.L., 2024. Analysis of fluid flow in fractal microfluidic channels. arXiv preprint arXiv:2409.12845.
29. Ba, Y., Mancenido, M.V. and Pan, R., 2024. Fill in the gaps: Model calibration and generalization with synthetic data. arXiv preprint arXiv:2410.10864.
30. Press, O., Zhang, M., Min, S., Schmidt, L., Smith, N.A. and Lewis, M., 2023, December. Measuring and narrowing the compositionality gap in language models. In Findings of the Association for Computational Linguistics: EMNLP 2023 (pp. 5687-5711).
31. Bengio, Y., Louradour, J., Collobert, R. and Weston, J., 2009, June. Curriculum learning. In Proceedings of the 26th annual international conference on machine learning (pp. 41-48).
32. Hu, J. and Wang, Z.L., 2025. Dynamic Wetting and Spreading of High-Viscosity Liquids on Grooved Substrates.
33. Hu, J. and Wang, Z. L., 2024. Crystallization morphology and self-assembly of polyacrylamide solutions during evaporation. arXiv preprint arXiv:2403.20191.
34. Hu, J. and Wang, Z.L., 2023. Inhibition of water vapor condensation by dipropylene glycol droplets on hydrophobic surfaces via vapor sink strategy. arXiv preprint arXiv:2311.03930.
35. Graves, A., Bellemare, M.G., Menick, J., Munos, R. and Kavukcuoglu, K., 2017, July. Automated curriculum learning for neural networks. In international conference on machine learning (pp. 1311-1320).
36. Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M.E. and Stone, P., 2020. Curriculum learning for reinforcement learning domains: A framework and survey. Journal of Machine Learning Research, 21(181), pp.1-50.
37. Wang, Z.L., Zhao, H., Xu, Z. and Hong, H., 2023. Suppression of water vapor condensation by glycerol droplets on hydrophobic surfaces. arXiv preprint arXiv:2311.03068.
38. Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N.A., Khashabi, D. and Hajishirzi, H., 2023, July. Self-instruct: Aligning language models with self-generated instructions. In Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers) (pp. 13484-13508).
39. Settles, B. 2009, Active Learning Literature Survey. *Univ. Wisconsin-Madison Tech. Rep.* 1648.
40. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
41. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J. and Awadallah, A.H., 2024, August. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In First Conference on Language Modeling.

 doi:10.20944/preprints202512.0499.v1

42. Karpas, E., Abend, O., Belinkov, Y., Lenz, B., Lieber, O., Ratner, N., Shoham, Y., Bata, H., Levine, Y., Leyton-Brown, K. and Muhlgay, D., 2022. MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. arXiv preprint arXiv:2205.00445.
43. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N. and Scialom, T., 2023. Toolformer: Language models can teach themselves to use tools. Advances in Neural Information Processing Systems, 36, pp.68539-68551.
44. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K.R. and Cao, Y., 2022, October. React: Synergizing reasoning and acting in language models. In The eleventh international conference on learning representations.
45. Chin, S.Y. and Why, D.N.K., Comparative of Multi-Agent System Frameworks: Crewai, Langchain, and Autogen. Langchain, and Autogen.
46. Dibia, V., Chen, J., Bansal, G., Syed, S., Fourney, A., Zhu, E., Wang, C. and Amershi, S., 2024. Autogen studio: A no-code developer tool for building and debugging multi-agent systems. arXiv preprint arXiv:2408.15247.
47. Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C. and Chen, C., 2022. Constitutional ai: Harmlessness from ai feedback. arXiv preprint arXiv:2212.08073.
48. Ganguli, D., Lovitt, L., Kernion, J., Askell, A., Bai, Y., Kadavath, S., Mann, B., Perez, E., Schiefer, N., Ndousse, K. and Jones, A., 2022. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. arXiv preprint arXiv:2209.07858.