

---

# Post-training is (Massive) Supervised Learning

Michael Hassid<sup>1,2</sup>, Yossi Adi<sup>1,2</sup>, Roy Schwartz<sup>2</sup>

<sup>1</sup>FAIR, Meta AI    <sup>2</sup>The Hebrew University of Jerusalem

michael.hassid@mail.huji.ac.il

## Abstract

The prevailing paradigm for training LLMs has evolved to rely on a massive post-training phase consisting of SFT and RL. In this position paper, we argue that this methodology effectively marks a reversion to the “pre-train then fine-tune” approach of the BERT era, explicitly tailoring models to the desired behaviors and specific benchmarks on which they are evaluated. We begin with a historical overview of LLMs, describing the different phases of the LLM evolution. We argue that the current landscape is remarkably similar to the early days of LLMs, where task performance heavily relied on fitting the models to in-distribution datasets. To empirically demonstrate this, we compare pre-trained models to randomly initialized ones, by fine-tuning both variants on modern reasoning datasets and evaluating them on competitive math and code benchmarks. We show that models post-trained from scratch yield highly non-trivial performance. Our findings suggest that current post-training methodologies function primarily as a distribution-fitting mechanism. We finish by positing that developing generally capable models and systems requires moving beyond extensive post-training for predefined behaviors, shifting instead toward training procedures where models “learn how to learn”.

## 1 Introduction

Historically, the prevailing machine learning paradigm relied on a fully supervised approach - training models directly on data distributions that corresponded to target tasks. This methodology was overtaken by the self-supervised approach that became dominant with large language models (LLMs). Over the years, the LLMs training paradigm has evolved to incorporate a post-training stage, consisting of supervised fine-tuning (SFT) and reinforcement learning (RL). These steps were designed to align the model towards specific behaviors. Today, this phase is a massive part of the training pipeline, consuming up to hundreds of billions of tokens (Olmo Team, 2025; Yang et al., 2026; Qwen Team, 2026b).

In this position paper, we argue that the NLP field has come full circle (Figure 1). Today, the training of leading LLMs, where post-training has become such a critical component, has effectively transformed back into a form of supervised learning. This paradigm marks a reversion to the “pre-train then fine-tune” methodology characteristic of the BERT era (Devlin et al., 2019; Peters et al., 2018; Radford et al., 2018). Rather than pursuing true zero-shot generalization (Brown et al., 2020), the prevailing strategy has shifted back toward multi-task optimization, explicitly fitting models to the distributions on which they are evaluated.

We begin with a brief historical overview of LLMs. We start by charting their early days as small-scale backbones for supervised learning models (Devlin et al., 2019; Liu et al., 2019). We then trace the shift toward massive pre-training scale, where models began achieving high performance via in-context learning with only a handful of inference-time examples (few-shot learning; Brown et al., 2020). Next, we describe the evolution of the post-training phase: what began as a short stage aimed at aligning a model’s behavior and formatting has expanded into a massive phase using hundreds of billions of tokens for SFT and RL trainings. We follow with a short discussion of the differences between the

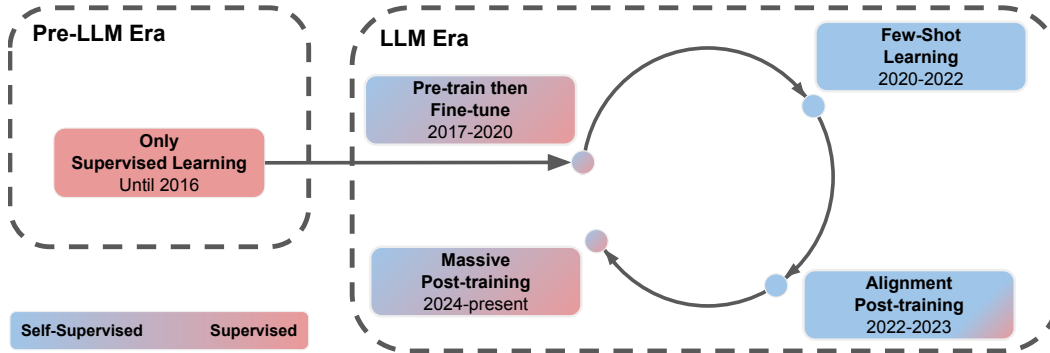


Figure 1: Evolution of LLM training paradigms. The field has progressed from only supervised learning (pre-LLM era), through several LLM-based stages: “pre-train then fine-tune”, few-shot learning, and post-training for alignment. We argue that the current era of massive post-training brings the field (almost) full circle. By heavily shifting model distributions toward specific benchmarks and desired behaviors, modern post-training effectively reverts to the supervised, distribution-fitting, mechanisms of earlier LLM eras.

two approaches. We conclude by arguing that, by systematically shifting model distributions toward specific benchmarks and desired behaviors, current methodologies effectively transform evaluation distributions into explicit training distributions.

To empirically demonstrate this observation, we design a series of experiments isolating the impact of large-scale post-training (specifically SFT) from traditional pre-training. We posit that if post-training merely unlocks or aligns pre-existing knowledge acquired during pre-training, a model trained from scratch strictly on SFT data should fail on reasoning tasks. Conversely, if it acts as an aggressive distribution-fitting mechanism, the non-pretrained model should achieve non-trivial performance.

We evaluate models across varying parameter scales on mathematical reasoning and competitive programming benchmarks. Remarkably, our results validate the latter scenario, demonstrating that post-training enables models to achieve surprisingly high results even in the complete absence of traditional pre-training. We find that models trained entirely from scratch, using SFT datasets only and without any prior linguistic or domain exposure, manage to achieve non-negligible results on highly difficult benchmarks. E.g., a 14B model scores 51% on LCB V5 (Jain et al., 2024) and a 7B model obtains 43% on AIME-24 (Balunović et al., 2025). Importantly, these models underperform their pre-trained counterparts by a margin of 8–20%, gaps that mirror the BERT and GPT-1 era, where pre-trained models held a similar advantage over supervised-only baselines. Furthermore, we show that a minimal pre-training setup of less than 0.1% of the original pre-training compute bridges up to half of the remaining performance deficit between non-pretrained and fully pre-trained models. Ultimately, this suggests modern post-training depends far less on pre-training’s emergent capabilities than previously thought, as high performance on desired benchmarks can rely on pulling targeted tasks directly into the model’s distribution.

We conclude that current LLM development relies on a massive, yet fundamentally reactive, post-training paradigm. Because this approach is inherently reactive, the field is trapped chasing the ever-evolving demands of LLM applications. We suggest that the path to generally capable models requires moving beyond static post-training for predefined benchmarks. We advocate for a shift in the training procedures, and suggest that they should include meta-learning where models *learn how to learn*. This will lead toward models and agentic systems designed to inherently self-evolve to solve novel, out-of-distribution tasks (Li et al., 2024b; Wu et al., 2025; Zhang et al., 2025a; Fang et al., 2025; Gao et al., 2025).

As a position paper, our contributions are not a proposal of a new method or training paradigm. Instead, our goal is to provide a critical perspective on the current state of LLM development, particularly the growing role of post-training in shaping model capabilities. Through both historical analysis and empirical observations, we argue that modern prac-

---

tices increasingly resemble traditional supervised learning at scale and distribution-fitting approaches. We highlight the broader implications of this trend for the field, and propose a high level direction for future research.

## 2 Back to Supervised Learning

We start by reviewing the history of LLMs, from their early days as backbones for fine-tuning to the modern, massive, post-training pipelines that characterize current models. (Section 2.1). Next, we discuss the differences between modern post-training methods and traditional ones (Section 2.2).

### 2.1 Pre-training and Post-training in LLMs: A Historical Overview

**From supervised learning to pre-train then fine-tune.** The prevailing approach prior to the current era of LLMs involved a straightforward supervised learning pipeline (Eisner, 1996; Klein & Manning, 2003; Shalev-Shwartz & Ben-David, 2014; Chen & Manning, 2014; Wang et al., 2016; Seo et al., 2016, *inter alia*). To tackle a specific task, researchers would first gather data and create a labeled dataset, then partition it into training and testing subsets. A model was subsequently trained on the training data, and its performance was evaluated on the validation and test sets. These evaluation sets were inherently highly in-distribution, meaning they were highly similar to the training data.

The introduction of models like ELMO, BERT, and GPT-1 marked a paradigm shift with the advent of the modern *pre-training* phase (Peters et al., 2018; Devlin et al., 2019; Radford et al., 2018). This new methodology involved first training the model in a self-supervised manner on large volumes of general text data, using techniques such as (masked) language modeling. These pre-trained models were subsequently fine-tuned on task-specific training sets and evaluated on their corresponding test sets (Wang et al., 2018; 2019). This fine-tuning approach yielded performance improvements of up to 20% over their supervised-only counterparts (Peters et al., 2018; Radford et al., 2018). This “pre-train then fine-tune” approach quickly became the new standard for training LLMs (Liu et al., 2019; Clark et al., 2020; Raffel et al., 2020), and was also adapted for other domains like vision and audio (Liu et al., 2022; Mohamed et al., 2022; Gui et al., 2024).

**Scaling and few-shot learning.** Subsequent research focused on progressively increasing the scale of this pre-training phase, eventually making it the sole dominant component of the training process. A notable example is GPT-3 (Brown et al., 2020), which was exclusively pre-trained and demonstrated advanced few-shot capabilities. This showed that models could achieve strong performance on benchmarks even without being explicitly trained on those specific data distributions. This approach became the de-facto approach for the following couple of years (Lieber et al., 2021; Zhang et al., 2022; BigScience et al., 2022).

**Post-training as a tool for model alignment.** Although large-scale pre-trained models demonstrated impressive capabilities on leading benchmarks, they often failed to follow explicit instructions or exhibit human-desired behaviors. To bridge this alignment gap, a post-training stage based on SFT and RL was established. Using a relatively small amount of curated data, this phase aligned a general-purpose base model to instruction and chat formats (Wei et al., 2021; Ouyang et al., 2022), as well as to human preferences (Bai et al., 2022). This preference alignment is typically achieved through RL frameworks (such as RLHF; Christiano et al., 2017; Stiennon et al., 2020).

**New, hard benchmarks arise.** With the advancement of LLM performance, the research community started building harder and more complicated benchmarks, with the goal of pushing the boundaries of the LLM technology. Early benchmarks such as GSM8k for mathematics and HumanEval for coding (Cobbe et al., 2021; Chen et al., 2021), targeted tasks that were challenging for models but relatively simple for humans. However, the ceiling for evaluation has since been raised by math competition benchmarks (Balunović et al., 2025), like AIME and HMMT, live code bench (LCB; Jain et al., 2024), which tests

---

competitive programming, and SWE/Terminal-bench which tests comprehensive software engineering skills (Jimenez et al., 2024; Deng et al., 2025; Ma et al., 2025; Yang et al., 2025b; Press et al., 2025; Merrill et al., 2026). These newer benchmarks test LLMs on tasks requiring human expert capabilities.

**The LLM community response: Post-training.** As the difficulty of benchmarks increased, the LLM community responded by tailoring models to these specific performance metrics (Anthropic, 2024; Singh et al., 2025; Kimi Team, 2026; Yang et al., 2026; Cursor Team, 2026). This has led to a post-training phase where the model’s data distribution is intentionally shifted to align with the desired output distribution, shaped by both the benchmark requirements and human preference data (Pi et al., 2026; Fu et al., 2026).

Consequently, post-training has evolved into a pivotal stage of the development lifecycle. This phase typically employs SFT datasets to bootstrap the RL process, targeting high-value tasks — such as software engineering (Jimenez et al., 2024; Deng et al., 2025), command-line control (Merrill et al., 2026), multi-hop web reasoning (Center for AI Safety et al., 2026), and more (Rein et al., 2024; Barres et al., 2025; Chollet et al., 2025). Scaling these post-training processes has emerged as a dominant paradigm. SFT data volumes have expanded up to hundreds of billions of tokens, incorporating diverse sources such as OMR, OCR, Nemotron-Math, Terminal-Corpus and more (Moshkov et al., 2025; Ahmad et al., 2025b;a; Du et al., 2025; Gandhi et al., 2025; Pi et al., 2026). This trajectory extends to the subsequent RL phase, which now scales to thousands of RL environments and hundreds of billions of interaction tokens (Yang et al., 2025a; Olmo Team, 2025; Badertdinov et al., 2025; Yang et al., 2026; Fu et al., 2026). Notably, current post-training pipelines now process data at a scale equivalent to the entire pre-training phase of models from just four to five years ago, such as GPT-3 and OPT (Brown et al., 2020; Zhang et al., 2022).

**Back to in-distribution specialization.** We argue that the growth of post-training suggests a skepticism regarding the few-shot generalization capabilities of pre-training alone (Brown et al., 2020). By systematically shifting model distributions toward specific benchmarks and desired behaviors, current methodologies effectively transform the evaluation distributions into explicit training distribution (Spiesberger et al., 2026).

In that sense, the NLP field has come full circle (Figure 1). Current LLMs are reminiscent of early LLMs like GPT-1 and BERT: we have pre-trained models that we fine-tune using supervised learning over specific distributions, which are the relevant distributions on which we evaluate them. This trajectory effectively marks a reversion to the “pre-train then fine-tune” methodology. Analogous to the BERT era, the prevailing strategy has shifted away from zero-shot generalization and back toward specific multi-task optimization, where models are explicitly fitted to the specific distributions on which they are evaluated and *currently* used. While these strategies may generalize to distributions similar to the training set, they often fail to generalize to underrepresented or significantly divergent distributions (Sharma & Chopra, 2026).

This methodology has established a recurring cycle. When a new capability is required, a benchmark is introduced, typically one where initial model performance is much worse compared to humans. Yet, these benchmarks are often saturated within a short time (a year or so). This rapid progress is frequently driven by the post-training process, where the benchmark is effectively pulled in-distribution via the curation of analogous SFT datasets and the construction of proxy RL environments.

## 2.2 Traditional vs. Modern Post-training

While the current LLM development pipeline marks a philosophical reversion to the “pre-train then fine-tune” methodology characteristic of the BERT era, critical distinctions between the two paradigms remain. Below, we discuss several key aspects of these differences.

Historically, the traditional post-training (or fine-tuning) phase was a relatively short stage. It used minimal data compared to the pre-training phase, serving merely to unlock capabilities or adapt a backbone to a specific format. During the BERT era, it was mainly used

---

for training on the specific training examples of each dataset (Devlin et al., 2019; Liu et al., 2019), while in the subsequent instruction-tuning era, it used millions of tokens (Ouyang et al., 2022; Touvron et al., 2023). In contrast, modern post-training has expanded into a massive phase. Current pipelines process diverse data sources at a scale of hundreds of billions of tokens for SFT and extensive RL (Qwen Team, 2025; Kimi Team, 2026; Copet et al., 2025; Yang et al., 2026). Notably, this phase now processes data volumes equivalent to the entire pre-training runs of leading models from just four to five years ago, such as GPT-3 and OPT (Brown et al., 2020; Zhang et al., 2022).

Furthermore, these traditional and modern paradigms differ in their data-generation mechanisms. Traditional post-training relies exclusively on static, task-specific datasets (Wang et al., 2018; 2019). In contrast, modern post-training employs a more dynamic approach. The SFT phase remains analogous to traditional supervised learning, optimizing over explicit input-output pairs (often distilled from other models). However, the RL phase shifts the supervisory signal exclusively to the terminal output (e.g., via a reward model); consequently, the intermediate training trajectories are generated dynamically by the model actively undergoing training.

A related mechanical distinction lies in the formulation of the training objective. Traditional fine-tuning framed downstream evaluations as classification or extraction tasks, typically applying the loss over a single output token—such as a dedicated classification head (Wang et al., 2018; 2019). Modern post-training, by contrast, is fundamentally generative. During SFT and RL, the model optimizes the next-token prediction loss over the entire output trajectory.

Importantly, similarly to the traditional fine-tuning setup, the whole modern post-training paradigm can be categorized as a supervised learning phase. Given that both SFT and RL ultimately rely on annotated data, the former through direct label mapping and the latter through the evaluation of model outputs, the underlying mechanics of this entire phase are fundamentally supervised.

Another difference lies in the relationship between training and test datasets. In the classic supervised pipeline, where each task was evaluated individually, a dataset was first collected and then randomly partitioned into training and test subsets. Because they originated from the exact same source, the test set was inherently highly in-distribution. Today, this dynamic is entirely inverted. The modern cycle begins with the introduction of a novel, highly difficult benchmark (the test set), where initial model performance is poor. To solve it, the LLM community systematically mines, synthesizes, and curates analogous SFT datasets and proxy RL environments designed to closely resemble the evaluation criteria (Moshkov et al., 2025; Ahmad et al., 2025b; Yang et al., 2026). Despite these differences, the result is strikingly similar: in both setups the training distribution closely resembles the test distribution.

Finally, a prominent distinction between modern post-training and the traditional approach is that the latter typically focused on a single task. In contrast, modern post-training objectives aim for the development of unified, generalist models by leveraging *multi-task* environments. This simultaneous exposure to various data distributions is intended to promote broader generalization beyond the explicit post-training data. For example, a model post-trained on competitive programming in a primary language is expected to generalize to alternative languages, assuming those secondary languages maintain some representation within the broader post-training corpus.

In the following section, we show that modern post-training can yield highly non-negligible results completely independent of pre-training. This highlights just how heavily current post-training pipelines shift model distributions to fit specific tasks.

### 3 Massive SFT can do without Pre-training

In this section, we investigate the efficacy of large-scale post-training in isolation. We empirically demonstrate that training LLMs on current SFT datasets yields surprisingly high in-distribution performance on hard math and code benchmarks, even in the absence of pre-training. This finding suggests that current post-training methods heavily optimize

---

models to excel on specific benchmarks, and that certain desired behaviors may rely less on the pre-training phase than previously thought.

We outline our experimental setup in Section 3.1. Next, we detail our findings in Section 3.2. Finally, we discuss our experimental scope and limitations in Section 3.3.

### 3.1 Experimental Setup

To understand the models’ performance of pre-trained vs. scratch initialization using SFT on desired distributions, we focus on competitive code and math benchmarks. We train models with a range of sizes and compare the results between the pre-trained versions to their from-scratch counterparts.

We train both variants on two distinct, large-scale SFT reasoning datasets. The math models are trained on the Open Math Reasoning dataset (OMR; Moshkov et al., 2025), consisting of  $\sim 3.2$ M question-answer pairs. The code models are trained on the Open Code Reasoning II dataset (OCR-II; Ahmad et al., 2025a), which contains about 1.4M Python and 1.2M C++ question-answer pairs.

We compare the different models’ performance on highly tracked math and competitive programming benchmarks to understand their reasoning capabilities — which is a primary focus and desired proficiency of recent model releases (Gemini Team, 2026; Qwen Team, 2026a; Yang et al., 2026). Our math evaluation uses MATH-500 (Lightman et al., 2023), and four competition math benchmarks from the MathArena leaderboard which are used as the current standard for testing leading math reasoning models (Balunović et al., 2025): AIME 24, AIME 25, HMMT February 25, and HMMT November 25. To evaluate reasoning for coding, we test the models over the LCB coding benchmark (Jain et al., 2024), using the V5 and V6 splits. We provide further details about the benchmarks in Appendix A.

To isolate the impact of pre-training on specific target distributions - in our case, reasoning math and code - we focus on models that lack explicit reasoning data (i.e., thinking trajectories) in their pre-training stage. Specifically, we use the Qwen-2.5-Base family of models, which offers extensive training on 18 trillion tokens, strong performance, and availability across multiple scales (Qwen Team, 2024). Our experiments use the 1.5B, 3B, 7B, and 14B parameter sizes.

We train all models on each of the SFT datasets for 50k steps, using a sequence length of 32k and a batch size of  $\sim 2$ M tokens (a total of  $\sim 100$ B tokens). Models trained from scratch use depth-scaled initialization (Zhang et al., 2019). For evaluation, we sample 10 answers per question (temperature = 0.6, top-p = 0.95) and report both pass@1 (average accuracy) and pass@5 (success rate within five attempts). See Appendix A for more details.

### 3.2 The Efficacy of SFT-only over Non-Pre-trained Models

We now turn to comparing pre-trained and non-pretrained models when heavily post-trained on specific distributions. If post-training merely unlocks pre-existing capabilities, the non-pretrained models should fail entirely as opposed to their pre-trained counterparts. In contrast, if it functions as an aggressive distribution-fitting mechanism, these non-pretrained models should achieve non-trivial performance.

We report the average over the math competition benchmarks alongside the other benchmarks in Table 1, while detailed per-benchmark math results are deferred to Table 2 in Appendix B. As expected, the Qwen-Base models, which do not undergo any post-training, exhibit poor performance across all benchmarks. While basic performance slightly increases with model size, the absolute accuracy remains low. These results are unsurprising given our use of a zero-shot evaluation paradigm, which is generally sub-optimal for base models on competitive benchmarks.

Applying the conventional SFT paradigm to these base models using OMR and OCR-II (100%-Pre-Math/Code) yields substantial performance increases across all benchmarks and scales. E.g., the 7B 100%-Pre-Math model surges to 93.6% on MATH-500 (an absolute gain of 89.4%) and 47.9% on the competition math average. This significant boost is

Size	Model	Math		Code		
		MATH-500 pass@1   5	Comp. Math pass@1   5	Model	LCB V5 pass@1   5	LCB V6 pass@1   5
1.5B	Base (no SFT)	2.4   10.0	0.0   0.0	Base (no SFT)	0.8   3.0	0.7   3.0
	0%-Pre-Math	74.4   87.6	24.7   43.4	0%-Pre-Code	35.0   52.1	33.6   48.8
	0.1%-Pre-Math	70.4   84.0	18.8   34.2	0.1%-Pre-Code	29.7   46.6	30.2   45.1
	100%-Pre-Math	90.1   96.8	38.8   61.4	100%-Pre-Code	49.2   64.7	47.4   61.2
3B	Base (no SFT)	4.1   14.8	0.5   2.1	Base (no SFT)	4.0   10.3	5.2   12.3
	0%-Pre-Math	74.8   85.5	24.9   42.3	0%-Pre-Code	41.5   59.4	40.4   55.9
	0.1%-Pre-Math	81.2   90.7	30.1   46.5	0.1%-Pre-Code	41.6   57.3	39.6   53.9
	100%-Pre-Math	92.9   97.6	45.9   62.3	100%-Pre-Code	55.6   69.2	53.7   66.4
7B	Base (no SFT)	4.2   16.3	0.2   0.8	Base (no SFT)	10.1   18.3	12.6   21.0
	0%-Pre-Math	79.5   89.2	27.1   48.9	0%-Pre-Code	44.4   62.2	42.9   57.4
	0.1%-Pre-Math	88.0   95.3	39.2   59.1	0.1%-Pre-Code	52.3   66.6	50.2   63.6
	100%-Pre-Math	93.6   97.6	47.9   65.6	100%-Pre-Code	58.7   71.2	56.0   67.8
14B	Base (no SFT)	13.6   38.8	0.7   2.4	Base (no SFT)	5.8   14.9	6.1   14.8
	0%-Pre-Math	79.2   89.9	29.0   48.5	0%-Pre-Code	51.0   65.9	48.5   62.9
	0.1%-Pre-Math	88.0   94.9	42.2   59.2	0.1%-Pre-Code	53.9   69.5	52.0   65.0
	100%-Pre-Math	92.5   96.8	44.8   60.3	100%-Pre-Code	60.3   72.6	57.8   69.6

Table 1: Results for the different models over the math and code benchmarks. The 0%-Pre models show non-negligible results, effectively beating the base models while slightly underperforming their 100%-Pre counterparts. The 0.1%-Pre models halves the gap.

anticipated, aligning with established trends in current LLM training methodologies (Copet et al., 2025; Moshkov et al., 2025; Ahmad et al., 2025b).

However, when considering the from-scratch models (0%-Pre-Math/Code), we observe that despite starting with randomly initialized weights and completely lacking prior exposure to language, mathematics, or code, they achieve surprisingly competitive results by relying entirely on the in-domain SFT distribution. Across all sizes, the models trained from scratch significantly outperform the standard base models, lagging behind their fully pre-trained counterparts by a margin of only 8% to 20%. For instance, the 14B 0%-Pre-Code model achieves 51% pass@1 on LCB V5, compared to the base model’s 6%, and to the 60% of the 100%-Pre-Code model with the same size. This is also evident when examining the math models, as can be seen for the 1.5B 0%-Pre-Math model that lags behind its fully pre-trained counterpart by only 10–20%, while achieving impressive results (74/25% at MATH-500/competition-math-average). These gaps mirror the GPT-1 and BERT era (Radford et al., 2018; Devlin et al., 2019), where pre-trained models outperformed their supervised-only counterparts by similar margins.

We provide thinking trajectories and answers for the 0%-Pre models in Appendix C. As shown, these non-pretrained models generate coherent, highly accurate reasoning chains which often produce correct answers for highly tracked benchmarks. Their ability to succeed in competitive programming and competition math solely through the SFT phase highlights how heavily post-training dominates task-specific performance.

**Cross-domain generalization.** A key question that arises is whether the post-training setup enables cross-domain generalization. E.g., whether the intensive post-training on math might yield superior performance on code. To test this hypothesis, we also conduct a cross-domain evaluation on the examined reasoning benchmarks. The results are presented in Table 3 in Appendix D. As expected, having seen no data from the alternate distribution, the 0%-Pre models demonstrate no cross-domain generalization and perform poorly. Importantly, we observe similar failures in the fully pre-trained models (100%-Pre), indicating that optimizing for a specific distribution in isolation is insufficient for reasoning generalization.<sup>1</sup>

<sup>1</sup>We note that standard post-training approaches often combine datasets from multiple domains. Studying the generalization effect in such setups is an important topic for future work.

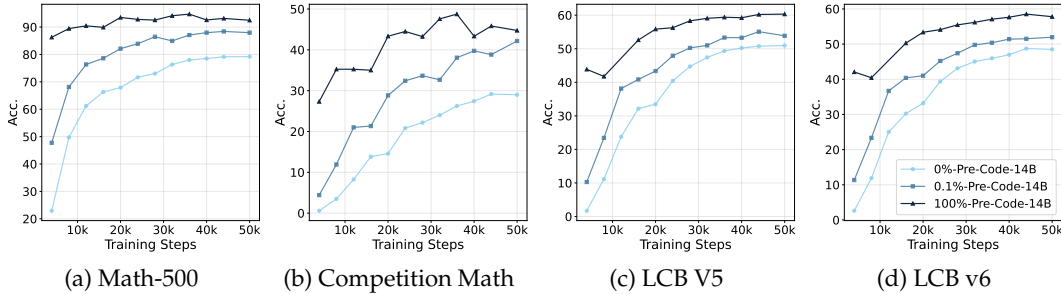


Figure 2: Evaluation curves during SFT using different amounts of pre-training backbones for the 14B scale. The 0/0.1%-Pre models show rapid improvement over the first 20k steps.

**Minimal pre-training setup.** To further understand the role of the pre-training phase for specific desired behaviors, we add for each model scale a *short pre-trained* version, where we first pre-train the model using general data on a fairly low amount of tokens (10B tokens, less than  $1/1,000$  of standard pre-training tokens compared to Qwen Team (2024)), then perform the SFT phase as for the other models. These models are trained on a limited sample of the DCLM dataset (Li et al., 2024a), before the SFT phase. See more details in Appendix A.

The results for the minimal pre-training phase (0.1%-Pre-Math/Code) are presented in Table 1 (with math competition results in Appendix B). Across the majority of models (excluding 1.5B 0.1%-Pre and 3B 0.1%-Pre-Code), this minimal initialization phase successfully closes 20% to 50% of the gap between from-scratch and full pre-training. 0.1%-Pre-Code at the 7B scale, for example, reaches 49.3% pass@1 on LCB V6, halving the distance between the from-scratch model (43%) and the fully pre-trained one (56%). We hypothesize that these gains stem from improved weight initialization and basic natural language acquisition, rather than exposure to domain-specific data (which is not explicitly included in the short pre-training phase). This suggests that the major part of the performance gap between pre-trained models and their non-pretrained counterparts is largely structural, serving as a better starting point for the SFT phase instead of adding specific knowledge.

**Performance during training.** Finally, we explore the performance of the different models over the course of training. Figure 2 presents the performance of the 14B model as training progresses, while Figures 6 to 8 in Appendix E present the curves of other models (Appendix F presents detailed math competition curves). As can be seen, the 100%-Pre-Math/Code models adapt to the different distributions relatively fast. Although they continue to improve later in training, they achieve the majority of their performance gains halfway through the SFT phase. Conversely, the 0%-Pre and 0.1%-Pre models exhibit weaker performance during the first few thousand training steps (due to a lack of prior knowledge), but quickly catch up, showing rapid improvement over the first 20k steps. While this again confirms the utility of pre-training, it is highly interesting to observe how well and relatively fast models can perform in-distribution relying solely on supervised data.

### 3.3 Experimental Scope and Limitations

Our experimental scope is subject to two primary limitations. First, our experiments are restricted to reasoning data: competitive math and code. We focused on these areas because they represent a primary focus of the current LLM field (Gemini Team, 2026; Qwen Team, 2026a; Yang et al., 2026) and benefit from the availability of large-scale, open-source SFT datasets (Ahmad et al., 2025a; Moshkov et al., 2025; Ahmad et al., 2025b). Second, our analysis focuses exclusively on SFT, whereas modern post-training pipelines typically include a subsequent RL phase.

Fundamentally, RL requires a pre-training stage for the model’s capacity to generate correct solutions to be rewarded on. Coupled with the fact that our pre-trained models consistently outperform their non-pretrained counterparts, we highlight that pre-training remains critical. Rather, our findings highlight the extent to which current post-training optimizes for specific

---

target distributions, enabling even models initialized from scratch to achieve surprising results on competitive benchmarks.

## 4 Learning how to Learn

As discussed in Section 2, the evolution of LLMs has transitioned from pre-trained backbones used for task-specific SFT to a massive post-training paradigm that aligns desired behaviors into the model’s distribution. Yet, this current pipeline remains fundamentally reactive: the post-training process is inherently restricted to specific use cases and behaviors that are *already known*, and for which large-scale post-training can be performed. These models demonstrate robust generalization within their established training distributions. However, adapting them to entirely novel behaviors necessitates full re-(post-)training cycles, a limitation that contrasts with the promise of few-shot learning (Brown et al., 2020).

This limitation drives the current trend of iterative model releases built upon the same pre-trained backbone, distinguished only by enhanced post-training setups that incorporate newly desired capabilities (e.g., the incremental progressions of GPT-5.x or Claude-4.x). Under the current paradigm, acquiring a novel capability demands a costly, ground-up re-post-training effort: curating new SFT datasets, engineering new RL environments, and executing continuous model updates just to bridge the newly identified capability gap. Proponents of the current paradigm may argue that scaling post-training data and compute can lead to achieving broader capabilities. Yet, this relies on the assumption that scaling predefined data and environments can efficiently keep pace with the ever-evolving demands placed on LLM systems. Therefore, we posit that the field must advance beyond these static, predefined post-training procedures toward a new evolutionary stage.

We argue for the development of significantly more robust systems. In our vision, models must inherently *“learn how to learn”*. Under this proposed paradigm, when an LLM system encounters a problem it cannot solve to the required degree of accuracy, it should autonomously adapt and evolve its capabilities to overcome that limitation.

One approach to realizing this vision is through autonomous, self-adapting LLM agents. We envision an agentic system capable of dynamically modifying its own prompts, underlying code, and even its model weights. When presented with a novel task where superficial adjustments (such as prompt engineering or modifying code execution) are insufficient, the system could autonomously replicate the LLM post-training pipeline. Specifically, it could search for and curate SFT data, engineer custom RL environments, and subsequently retrain its internal models. This method can include attempting a solution, while iteratively learning through any failures. The system would then evaluate the updated model’s performance on the target task, iterating on this process as necessary. Furthermore, under resource constraints, such a system could intelligently opt to train a smaller, highly specialized model to serve as a tool for future instances of the new task.

Of course, the acquisition of the above meta-learning capabilities must be integrated as a core objective within modern training paradigms. Just as traditional pre-training serves as the foundational phase for general knowledge and linguistic skills, this proposed meta-learning phase acts as the mechanism for dynamic, post-deployment task acquisition (similar to the promise of few-shot learning). Ultimately, this establishes a new level of machine intelligence defined by the ability to *“learn how to learn”*.

Crucially, this foundational training would ideally only be required once. Once a model acquires the capacity to self-evolve, it will be able to theoretically improve indefinitely, limited only by computational resources. Consequently, this paradigm will relieve researchers from continuously engineering and maintaining post-training procedures by delegating the control of ongoing adaptation directly to the models themselves.

We note that our suggested paradigm idea is not limited to the specific mechanism described above, and can have several forms and generalizability (Bandel et al., 2026). Various approaches can successfully enable systems to learn how to solve unknown tasks (Li et al., 2024b; Zhang et al., 2025a;b; Wang et al., 2026; Wu et al., 2025; Fang et al., 2025; Gao et al., 2025; Zhang et al., 2026; Weng et al., 2026; Xia et al., 2026). Achieving this self-evolving

---

capacity may also require moving beyond standard Transformer backbones to alternative architectures that yield highly effective in-context learning for novel tasks without massive supervised learning (Zhang & Bottou, 2025). The unifying idea for optimizing all such systems is the shared principle of “learning how to learn”. Different approaches to this objective will naturally result in varying systems, which will result in a robust ecosystem of self-improving models.

## 5 Conclusion

In this position paper, through a historical overview of LLMs, we argued that current training paradigms effectively revert to the “pre-train then fine-tune” methodology of the BERT era. Under this approach, models are explicitly tailored to desired behaviors and specific evaluation benchmarks. We empirically demonstrated this by showing that models post-trained from scratch yield highly non-trivial performance, suggesting that current post-training methodologies function primarily as a distribution-fitting mechanism. Finally, we argued that prevailing post-training approaches are fundamentally reactive. To achieve more generalized capabilities, we advocate for a shift toward meta-learning paradigms, where models “learn how to learn”.

## Acknowledgments

We thank Ori Yoran and Gallil Maimon for countless discussions, brainstorming sessions, and their valuable perspectives on this work. We also thank Miri Varshavsky Hassid for the great feedback and moral support.

## References

- Wasi Uddin Ahmad, Somshubra Majumdar, Aleksander Ficek, Sean Narenthiran, Mehrzad Samadi, Jocelyn Huang, Siddhartha Jain, Vahid Noroozi, and Boris Ginsburg. Opencodereasoning-ii: A simple test time scaling approach via self-critique. *arXiv preprint arXiv:2507.09075*, 2025a.
- Wasi Uddin Ahmad, Sean Narenthiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. Opencodereasoning: Advancing data distillation for competitive coding, 2025b. URL <https://arxiv.org/abs/2504.01943>. arXiv:2504.01943.
- Anthropic. Claude 3.5 sonnet model card addendum. 2024. URL [https://www-cdn.anthropic.com/fed9cc193a14b84131812372d8d5857f8f304c52/Model\\_Card\\_Claude\\_3\\_Addendum.pdf](https://www-cdn.anthropic.com/fed9cc193a14b84131812372d8d5857f8f304c52/Model_Card_Claude_3_Addendum.pdf).
- Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *arXiv preprint arXiv:2505.20411*, 2025.
- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.
- Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmark*, 2025.
- Elron Bandel, Asaf Yehudai, Alexandre Lacoste, Avijit Ghosh, Graham Neubig, Margaret Mitchell, Michal Shmueli-Scheuer, and Leshem Choshen. Position: Agentic systems should be general. *SSRN Electronic Journal*, 2026. doi: 10.2139/ssrn.6176178. URL <https://ssrn.com/abstract=6176178>. Posted: 18 Feb 2026.

- 
- Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan.  $\tau^2$ -bench: Evaluating conversational agents in a dual-control environment, 2025. URL <https://arxiv.org/abs/2506.07982>.
- BigScience, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- Center for AI Safety, Scale AI, and HLE Contributors Consortium. A benchmark of expert-level academic questions to assess AI capabilities. *Nature*, 649:1139–1146, 2026. doi: 10.1038/s41586-025-09962-4. URL <https://arxiv.org/abs/2501.14249>.
- Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In Alessandro Moschitti, Bo Pang, and Walter Daelemans (eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 740–750, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1082. URL <https://aclanthology.org/D14-1082/>.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebggen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. arXiv2107.03374.
- Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. Arcagi-2: A new challenge for frontier ai reasoning systems. *arXiv preprint arXiv:2505.11831*, 2025.
- Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.
- Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- Jade Copet, Quentin Carbonneaux, Gal Cohen, Jonas Gehring, Jacob Kahn, Jannik Kossen, Felix Kreuk, Emily McMilin, Michel Meyer, Yuxiang Wei, et al. Cwm: An open-weights llm for research on code generation with world models. *arXiv preprint arXiv:2510.02387*, 2025.
- Cursor Team. Composer 2 technical report. *arXiv preprint arXiv:2603.24477*, 2026.

- 
- Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.
- Wei Du, Shubham Toshniwal, Branislav Kisacanin, Sadegh Mahdavi, Ivan Moshkov, George Armstrong, Stephen Ge, Edgar Minasyan, Feng Chen, and Igor Gitman. Nemotron-math: Efficient long-context distillation of mathematical reasoning from multi-mode supervision. *arXiv preprint arXiv:2512.15489*, 2025.
- Jason Eisner. Three new probabilistic models for dependency parsing: An exploration. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, 1996.
- Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, et al. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025.
- Dayuan Fu, Shenyu Wu, Yunze Wu, Zerui Peng, Yaxing Huang, Jie Sun, Ji Zeng, Mohan Jiang, Lin Zhang, Yukun Li, Jiarui Hu, Liming Liu, Jinlong Hou, and Pengfei Liu. davinci-env: Open swe environment synthesis at scale, 2026. URL <https://arxiv.org/abs/2603.13023>.
- Kanishk Gandhi, Shivam Garg, Noah D. Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling rl environments for terminal agents. *arXiv preprint arXiv:2601.16443*, 2025.
- Huan Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian, Zhenghailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: On path to artificial super intelligence, 2025. URL <https://arxiv.org/abs/2507.21046>.
- Gemini Team. Gemini 3.1 pro: A smarter model for your most complex tasks, 2026. URL <https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/>.
- Jie Gui, Tuo Chen, Jing Zhang, Qiong Cao, Zhenan Sun, Hao Luo, and Dacheng Tao. A survey on self-supervised learning: Algorithms, applications, and future trends. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9052–9071, 2024.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=VTF8yNQM66>.

- 
- Kimi Team. Kimi k2.5: Visual agentic intelligence, 2026. URL <https://arxiv.org/abs/2602.02276>.
- Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pp. 423–430, Sapporo, Japan, July 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075150. URL <https://aclanthology.org/P03-1054/>.
- Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruva Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. Datacomp-lm: In search of the next generation of training sets for language models. *arXiv preprint arXiv:2406.11794*, 2024a.
- Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.
- Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.
- Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. Autoflow: Automated workflow generation for large language model agents. *arXiv preprint arXiv:2407.12821*, 2024b.
- Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. Jurassic-1: Technical details and evaluation. *White Paper. AI21 Labs*, 1(9):1–17, 2021.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.
- Shuo Liu, Adria Mallol-Ragolta, Emilia Parada-Cabaleiro, Kun Qian, Xin Jing, Alexander Kathan, Bin Hu, and Bjoern W Schuller. Audio self-supervised learning: A survey. *Patterns*, 3(12), 2022.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
- Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- Jeffrey Jian Ma, Milad Hashemi, Amir Yazdanbakhsh, Kevin Swersky, Ofir Press, Enhui Li, Vijay Janapa Reddi, and Parthasarathy Ranganathan. Swe-fficiency: Can language models optimize real-world repositories on real workloads? *arXiv preprint arXiv:2511.06090*, 2025.
- Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.
- Abdelrahman Mohamed, Hung-yi Lee, Lasse Borgholt, Jakob D Havtorn, Joakim Edin, Christian Igel, Katrin Kirchhoff, Shang-Wen Li, Karen Livescu, Lars Maaløe, et al. Self-supervised speech representation learning: A review. *IEEE Journal of Selected Topics in Signal Processing*, 16(6):1179–1210, 2022.

- 
- Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. *arXiv preprint arXiv:2504.16891*, 2025.
- Olmo Team. Olmo 3, 2025. *arXiv:2512.13961*.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. Codeforces. <https://huggingface.co/datasets/open-r1/codeforces>, 2025.
- Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In Marilyn Walker, Heng Ji, and Amanda Stent (eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL <https://aclanthology.org/N18-1202/>.
- Renjie Pi, Grace Lam, Mohammad Shoeybi, Pooya Jannaty, Bryan Catanzaro, and Wei Ping. On data engineering for scaling llm terminal capabilities. *arXiv preprint arXiv:2602.21193*, 2026.
- Ori Press, Brandon Amos, Haoyu Zhao, Yikai Wu, Samuel K Ainsworth, Dominik Krupke, Patrick Kidger, Touqir Sajed, Bartolomeo Stellato, Jisun Park, et al. Algotune: Can language models speed up general-purpose numerical programs? *arXiv preprint arXiv:2507.15887*, 2025.
- Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL <https://qwenlm.github.io/blog/qwen2.5/>.
- Qwen Team. Qwq-32b: Embracing the power of reinforcement learning, March 2025. URL <https://qwenlm.github.io/blog/qwq-32b/>.
- Qwen Team. Qwen3.5: Accelerating productivity with native multimodal agents, February 2026a. URL <https://qwen.ai/blog?id=qwen3.5>.
- Qwen Team. Qwen3-coder-next technical report. Technical report, 2026b. URL [https://github.com/QwenLM/Qwen3-Coder/blob/main/qwen3\\_coder\\_next\\_tech\\_report.pdf](https://github.com/QwenLM/Qwen3-Coder/blob/main/qwen3_coder_next_tech_report.pdf). Accessed: 2026-02-03.
- Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training, 2018.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First conference on language modeling*, 2024.
- Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.
- Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

- 
- Aman Sharma and Paras Chopra. Esolang-bench: Evaluating genuine reasoning in large language models via esoteric programming languages. *arXiv preprint arXiv:2603.09678*, 2026.
- Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.
- Ari Spiesberger, Juan J Vazquez, Nicky Pochinkov, Tomáš Gavenčíak, Peli Grietzer, Gavin Leech, and Nandi Schoots. Soft contamination means benchmarks test shallow generalization. *arXiv preprint arXiv:2602.12413*, 2026.
- Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Tal Linzen, Grzegorz Chrupała, and Afra Alishahi (eds.), *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL <https://aclanthology.org/W18-5446/>.
- Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.
- Wenyi Wang, Piotr Piekos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-g\`odel machine: Human-level coding agent development by an approximation of the optimal self-improving machine. In *The Fourteenth International Conference on Learning Representations*, 2026. URL <https://openreview.net/forum?id=T0EiEuh00L>.
- Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based LSTM for aspect-level sentiment classification. In Jian Su, Kevin Duh, and Xavier Carreras (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 606–615, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1058. URL <https://aclanthology.org/D16-1058/>.
- Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.
- Zhaotian Weng, Antonis Antoniadis, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. Group-evolving agents: Open-ended self-improvement via experience sharing. *arXiv preprint arXiv:2602.04837*, 2026.
- Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. Evolver: Self-evolving llm agents through an experience-driven lifecycle, 2025. URL <https://arxiv.org/abs/2510.16079>.
- Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.
- John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025a.

- 
- John Yang, Kilian Lieret, Joyce Yang, Carlos E Jimenez, Ofir Press, Ludwig Schmidt, and Diyi Yang. Codeclash: Benchmarking goal-oriented software engineering. *arXiv preprint arXiv:2511.00839*, 2025b.
- Zhuolin Yang, Zihan Liu, Yang Chen, Wenliang Dai, Boxin Wang, Sheng-Chieh Lin, Chankyu Lee, Yangyi Chen, Dongfu Jiang, Jiafan He, Renjie Pi, Grace Lam, Nayeon Lee, Alexander Bukharin, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nemotron-cascade 2: Post-training llms with cascade rl and multi-domain on-policy distillation, 2026. URL <https://arxiv.org/abs/2603.19220>. arXiv:2603.19220.
- Alex L Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025a.
- Biao Zhang, Ivan Titov, and Rico Sennrich. Improving deep transformer with depth-scaled initialization and merged attention. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 898–909, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1083. URL <https://aclanthology.org/D19-1083/>.
- Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025b.
- Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents. *arXiv preprint arXiv:2603.19461*, 2026.
- Jianyu Zhang and Leon Bottou. Memory mosaics at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL <https://openreview.net/forum?id=IfD2MKTmWv>.
- Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

---

## A Detailed Experimental Setup

**Evaluation benchmarks.** The MATH-500 comprises 500 high school math questions (Lightman et al., 2023). The competition math benchmarks, AIME 24, AIME 25, HMMT February 25, and HMMT November 25, taken from the MathArena leaderboard (Balunović et al., 2025). Each of them contains 30 competition level math problems. The LCB coding benchmark is built from competitive programming questions, where the V5 splits consists of 327 problems and the V6 split has 454 problems (Jain et al., 2024).

**SFT datasets.** The problems of the OMR dataset were collected from AoPS forums,<sup>2</sup> while the solutions were generated using DeepSeek-R1 and Qwen-QwQ (Guo et al., 2025; Qwen Team, 2025). As for OCR-II, it aggregates programming challenges from TACO (Li et al., 2023), APPS (Hendrycks et al., 2021), CodeContests (Li et al., 2022), and CodeForces (Penedo et al., 2025), pairing them with solutions that were generated by DeepSeek-R1 and subsequently verified by Qwen-QwQ.

**Short pre-training details.** The minimal pre-training phase is based on the DCLM dataset (Li et al., 2024a), which was constructed as a high-quality dataset for pre-training. This phase uses  $\sim 10\text{B}$  tokens, which is strictly less than  $1/1,000$  of the original pre-training corpus used by Qwen-2.5-Base. The short pre-training phase retains the configuration of the SFT phase below but employs a constant learning rate of  $1\text{e-}4$  and 500 steps for warm-up.

**Training and evaluation details.** We use the AdamW optimizer (Loshchilov & Hutter, 2017) with a peak learning rate of  $3\text{e-}4$ , a cosine decay scheduler, and a 2,000-step linear warmup. For evaluation, we use zero-shot prompt style, which evaluates the ability of models also to follow instructions. We detail our math and code prompts templates below.

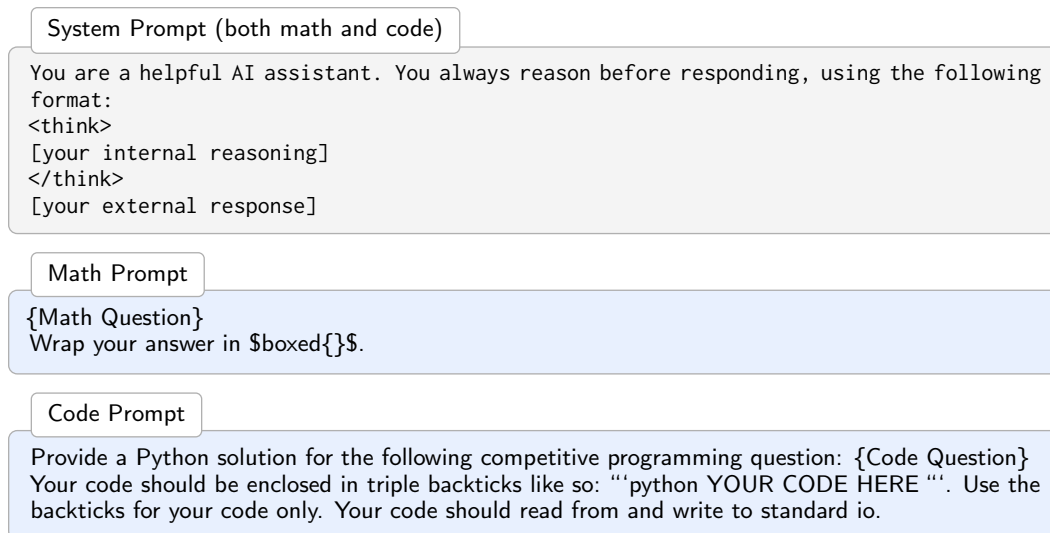


Figure 3: Prompt templates for the math and code questions.

---

<sup>2</sup>[artofproblemsolving.com/community](https://artofproblemsolving.com/community)

---

## B Full Competition Math Results

We present the benchmark-specific results for the competition math benchmarks: AIME-24, AIME-25, HMMT February 25 and HMMT November 25 in Table 2.

Size	Model	AIME'24	AIME'25	HMMT Feb	HMMT Nov
		pass@1   5	pass@1   5	pass@1   5	pass@1   5
1.5B	Base (no SFT)	0.0   0.0	0.0   0.0	0.0   0.0	0.0   0.0
	0%-Pre-Math	32.0   57.6	30.7   50.1	16.7   32.7	19.3   33.2
	0.1%-Pre-Math	26.7   51.1	21.7   34.5	10.7   24.4	16.0   26.6
	100%-Pre-Math	56.0   77.0	38.0   61.9	26.0   52.8	35.3   54.0
3B	Base (no SFT)	1.3   5.2	0.0   0.0	0.3   1.7	0.3   1.7
	0%-Pre-Math	40.7   60.9	26.7   46.4	10.7   20.0	21.7   42.0
	0.1%-Pre-Math	44.0   68.1	35.7   51.3	17.0   28.9	23.7   37.8
	100%-Pre-Math	62.7   79.1	47.0   64.1	33.0   50.3	41.0   55.6
7B	Base (no SFT)	0.3   1.7	0.0   0.0	0.0   0.0	0.3   1.7
	0%-Pre-Math	42.7   69.4	27.7   51.6	14.7   32.8	23.3   42.0
	0.1%-Pre-Math	50.7   73.7	44.3   63.0	28.7   49.5	33.0   50.3
	100%-Pre-Math	67.0   79.9	48.0   70.3	32.3   53.2	44.3   59.0
14B	Base (no SFT)	1.3   4.7	0.0   0.0	0.3   1.7	1.0   3.1
	0%-Pre-Math	41.7   64.3	30.3   46.6	18.7   38.1	25.3   45.2
	0.1%-Pre-Math	57.7   72.3	44.7   65.8	32.3   54.1	34.0   44.6
	100%-Pre-Math	60.3   75.1	47.0   61.8	31.3   49.1	40.3   55.3

Table 2: Results for the math models over the different math competition benchmarks.

## C Examples of Generated Outputs

Figure 4 presents an example response from the 0%-Pre-Math (14B) model for an AIME 25 question, while Figure 5 demonstrates the 0%-Pre-Code (7B) model's response to an LCB V6 question. As can be seen, the reasoning trajectory is coherent and understandable, although the model was trained without any prior linguistic knowledge.

Math Question

Find the sum of all integer bases  $b > 9$  for which  $17_b$  is a divisor of  $97_b$ . Wrap your answer in  $\boxed{\quad}$

0%-Pre-Math (14B) Response

`<think>` Okay, let's see. I need to find all integers  $b$  greater than 9 such that  $17_b$  divides  $97_b$ . Then sum those  $b$ 's. Hmm. First, let's convert both numbers from base  $b$  to base 10 to work with them more easily. The number 17 in base  $b$  is equal to  $1 \cdot b + 7$  in decimal. Similarly, 97 in base  $b$  is  $9 \cdot b + 7$  in decimal. So, we need  $(1 \cdot b + 7)$  divides  $(9 \cdot b + 7)$ . Let me write that as an equation:  $(9b + 7)$  divided by  $(b + 7)$  should be an integer. Let me compute  $(9b + 7) \div (b + 7)$ . Maybe perform the division to see what the quotient and remainder are. Divide  $9b + 7$  by  $b + 7$ . Let's see: How many times does  $(b + 7)$  go into  $9b + 7$ ? Let's do polynomial division. Divide  $9b$  by  $b$ , which is 9. Multiply  $(b + 7)$  by 9:  $9b + 63$ . Subtract that from  $9b + 7$ :  $(9b + 7) - (9b + 63) = -56$ . So the division gives 9 with a remainder of -56. Therefore,  $(9b + 7) = 9 \cdot (b + 7) - 56$ . So,  $(9b + 7) / (b + 7) = 9 - 56 / (b + 7)$ . For this to be an integer,  $56 / (b + 7)$  must be an integer. Therefore,  $(b + 7)$  must be a divisor of 56. But since  $b > 9$ , then  $b + 7 > 16$ . So, we need to find all divisors of 56 that are greater than 16, then subtract 7 to get  $b$ . First, list all positive divisors of 56. 56 factors into  $2^3 \cdot 7^1$ . So the divisors are 1, 2, 4, 7, 8, 14, 28, 56. From these, the divisors greater than 16 are 28 and 56. Therefore,  $b + 7 = 28 \implies b = 21$ , and  $b + 7 = 56 \implies b = 49$ . So the possible values of  $b$  are 21 and 49. Summing them gives  $21 + 49 = 70$ . Wait, but let me check. Let's verify each  $b$ . First,  $b = 21$ . Then  $17_{21}$  is  $1 \cdot 21 + 7 = 28$ .  $97_{21}$  is  $9 \cdot 21 + 7 = 189 + 7 = 196$ . Check if 28 divides 196.  $196 \div 28 = 7$ , which is an integer. So that's good. Next,  $b = 49$ .  $17_{49}$  is  $1 \cdot 49 + 7 = 56$ .  $97_{49}$  is  $9 \cdot 49 + 7 = 441 + 7 = 448$ .  $448 \div 56 = 8$ , which is also an integer. So that works too. Are there any other divisors of 56 greater than 16? Let's check the list again. The divisors are 1, 2, 4, 7, 8, 14, 28, 56. So after 14 comes 28 and 56. So yes, only 21 and 49. Therefore, the sum is 70. I think that's it. Let me make sure there's no mistake in the logic. The key step was realizing that  $(9b + 7)$  divided by  $(b + 7)$  must be integer, which requires that the remainder is zero. The remainder was -56, so  $(b + 7)$  must divide 56. Then considering the divisors of 56 greater than 16. That seems correct. So the answer should be  $21 + 49 = 70$ .`</think>`  
To solve the problem of finding the sum of all integer bases  $b > 9$  for which  $17_b$  divides  $97_b$ , we proceed as follows:  
1. **Convert the numbers from base  $b$  to base 10:** - The number  $17_b$  in base 10 is  $1 \cdot b + 7 = b + 7$ .  
- The number  $97_b$  in base 10 is  $9 \cdot b + 7 = 9b + 7$ .  
2. **Set up the divisibility condition:** - We need  $b + 7$  to divide  $9b + 7$ . This means  $\frac{9b + 7}{b + 7}$  must be an integer.  
3. **Perform the division:** - Divide  $9b + 7$  by  $b + 7$ :  
 $\frac{9b + 7}{b + 7} = 9 - \frac{56}{b + 7}$  - For this to be an integer,  $\frac{56}{b + 7}$  must be an integer. Therefore,  $b + 7$  must be a divisor of 56.  
4. **Identify the divisors of 56:** - The divisors of 56 are: 1, 2, 4, 7, 8, 14, 28, 56. - Since  $b > 9$ , we need  $b + 7 > 16$ . Thus, the relevant divisors are 28 and 56.  
5. **Determine the corresponding values of  $b$ :** - If  $b + 7 = 28$ , then  $b = 21$ . - If  $b + 7 = 56$ , then  $b = 49$ .  
6. **Verify the solutions:** - For  $b = 21$ :  $17_{21} = 1 \cdot 21 + 7 = 28$   $97_{21} = 9 \cdot 21 + 7 = 189 + 7 = 196$   $196 \div 28 = 7$  (an integer) - For  $b = 49$ :  $17_{49} = 1 \cdot 49 + 7 = 56$   $97_{49} = 9 \cdot 49 + 7 = 441 + 7 = 448$   $448 \div 56 = 8$  (an integer)  
7. **Sum the valid values of  $b$ :**  $21 + 49 = 70$   
Thus, the sum of all integer bases  $b > 9$  for which  $17_b$  divides  $97_b$  is  $\boxed{70}$ .

Figure 4: A response example of 0%-Pre-Math (14B) on an example from AIME 25.

### Code Question

Provide a Python solution for the following competitive programming question: There is a stack of  $N$  cards, and the  $i$ -th card from the top has an integer  $A_i$  written on it. You take  $K$  cards from the bottom of the stack and place them on top of the stack, maintaining their order. Print the integers written on the cards from top to bottom after the operation. Input The input is given from Standard Input in the following format: $N K A_1 A_2 \dots A_N$

Output

Let  $B_i$  be the integer written on the  $i$ -th card from the top of the stack after the operation. Print  $B_1, B_2,$

$\dots, B_N$  in this order, separated by spaces.

Constraints  $1 \leq K < N \leq 100$  ;  $1 \leq A_i \leq 100$ - All input values are integers.

{Example IO pairs are omitted}

Your code should be enclosed in triple backticks like so: “python YOUR CODE HERE “. Use the backticks for your code only. Your code should read from and write to standard io.

### 0%-Pre-Code (7B) Response

<think> Okay, let's see. The problem is about moving  $K$  cards from the bottom of a stack to the top, keeping their order. So, for example, if the original stack is  $[1,2,3,4,5]$  (with 1 being the top), and  $K$  is 3, then we take the bottom three cards  $(3,4,5)$  and put them on top. So the new stack becomes  $[3,4,5,1,2]$ . Then we need to output them top to bottom. Hmm. So the input is  $N$  and  $K$ , then the list  $A$  which is the initial stack from top to bottom. Wait, wait. Wait, the problem says the  $i$ -th card from the top has  $A_i$  written on it. So the first element in the input list is the top card, and the last is the bottom. So the initial stack is  $A_1$  (top),  $A_2, \dots, A_N$  (bottom). So when we take  $K$  cards from the bottom, which would be the last  $K$  elements of the list. Then we move those to the top. So the new stack is  $[A_{N-K+1}, A_{N-K+2}, \dots, A_N]$  followed by  $[A_1, A_2, \dots, A_{N-K}]$ . Wait, no. Wait, the original stack is  $A_1$  (top),  $A_2, \dots, A_N$  (bottom). So the bottom  $K$  cards are  $A_{N-K+1}, A_{N-K+2}, \dots, A_N$ . Because if  $N$  is 5 and  $K$  is 3, then the bottom 3 cards are positions 3,4,5 (since  $5-3+1=3$ ). Wait, no. Wait, if the stack is 1 (top), 2, 3,4,5 (bottom), then the bottom three cards are 3,4,5. So the indices in the list would be  $[2], [3], [4]$  (assuming zero-based). Wait, the list is given as  $A_1$  to  $A_N$ . So for the sample input 1, the list is  $[1,2,3,4,5]$ . The bottom three cards are 3,4,5. So their indices in the list are 2,3,4 (since Python is zero-based). So to get the bottom  $K$  elements, we need to take the last  $K$  elements of the list. So the process is: take the last  $K$  elements (from the bottom) and put them on top. So the new list is (last  $K$  elements) + (first  $N-K$  elements). Then, when we output, we print each element in order, which is the new top to bottom. So for sample input 1, the original list is  $[1,2,3,4,5]$ . Take last 3 elements  $[3,4,5]$ , then add the first 2 elements  $[1,2]$ . So new list is  $[3,4,5,1,2]$ . Then output as space-separated. So the steps are: 1. Read  $N$  and  $K$  from input. 2. Read the list  $A$  as integers. 3. Split the list into two parts: the last  $K$  elements (bottom  $K$ ) and the rest. 4. The new stack is the concatenation of the last  $K$  elements and the first  $N-K$  elements. 5. Print the new stack elements in order, separated by spaces. So in code: Read  $N, K = \text{map}(\text{int}, \text{input}().\text{split}())$   $A = \text{list}(\text{map}(\text{int}, \text{input}().\text{split}()))$   $\text{new\_stack} = A[-K:] + A[:-K]$   $\text{print}(' '.\text{join}(\text{map}(\text{str}, \text{new\_stack})))$ . Wait, but wait. Let's test with sample input 1.  $N=5, K=3$ .  $A$  is  $[1,2,3,4,5]$ .  $A[-3:]$  is  $[3,4,5]$ ,  $A[:-3]$  is  $[1,2]$ . So concatenating gives  $[3,4,5,1,2]$ . Which is correct. Sample input 2:  $N=6, K=2$ .  $A$  is  $[1,2,1,2,1,2]$ . The last two elements are  $[1,2]$ , the first 4 elements are  $[1,2,1,2]$ . So new stack is  $[1,2] + [1,2,1,2] \rightarrow [1,2,1,2,1,2]$ . So output is same as input, which matches the sample output. So the code seems correct. But wait, the problem says that after moving the  $K$  cards, the new stack's top is the first element. So the code's approach of taking the last  $K$  elements and putting them first (as the new top) is correct. Yes. So the code should work. </think>

```
python
# Read input
N, K = map(int, input().split())
A = list(map(int, input().split()))
# Calculate the new stack
new_stack = A[-K:] + A[:-K]
# Output the result
print(' '.join(map(str, new_stack)))
"
```

Figure 5: A response example of 0%-Pre-Math (7B) on an example from LCB V6.

## D Cross-Domain Results

We present the cross-domain results of the different models. Table 3 presents both the code models performance over the math benchmarks, and the results of the math models over the LCB benchmark.<sup>3</sup> As can be seen, for all cases, models fine-tuned on a specific distribution failed to generalize to another.

Size	Code models on math			Math models on code		
	Model	MATH-500 pass@1   5	Comp. Math pass@1   5	Model	LCB V5 pass@1   5	LCB V6 pass@1   5
1.5B	0%-Pre-Code	0.0   0.0	0.0   0.0	0%-Pre-Math	0.0   0.0	0.0   0.0
	100%-Pre-Code	0.0   0.0	0.0   0.0	100%-Pre-Math	0.0   0.0	0.0   0.2
3B	0%-Pre-Code	0.0   0.0	0.0   0.0	0%-Pre-Math	0.0   0.0	0.0   0.0
	100%-Pre-Code	0.0   0.0	0.0   0.0	100%-Pre-Math	0.1   0.3	0.4   1.1
7B	0%-Pre-Code	0.0   0.0	0.0   0.0	0%-Pre-Math	0.0   0.0	0.0   0.0
	100%-Pre-Code	0.1   0.3	0.0   0.0	100%-Pre-Math	0.0   0.0	0.2   0.4
14B	0%-Pre-Code	0.0   0.0	0.0   0.0	0%-Pre-Math	0.0   0.0	0.0   0.0
	100%-Pre-Code	0.1   0.7	0.0   0.0	100%-Pre-Math	0.0   0.2	0.0   0.0

Table 3: Results for code models on math benchmarks, and math models on code benchmarks.

<sup>3</sup>We do not detail the benchmark specific results for the competition math benchmarks as they are all 0.0.

## E Performance Curves

Figure 6, Figure 7 and Figure 8 presents the performance curves of the 7B/3B/1.5B models, respectively, using different amounts of data for the pre-training backbones.

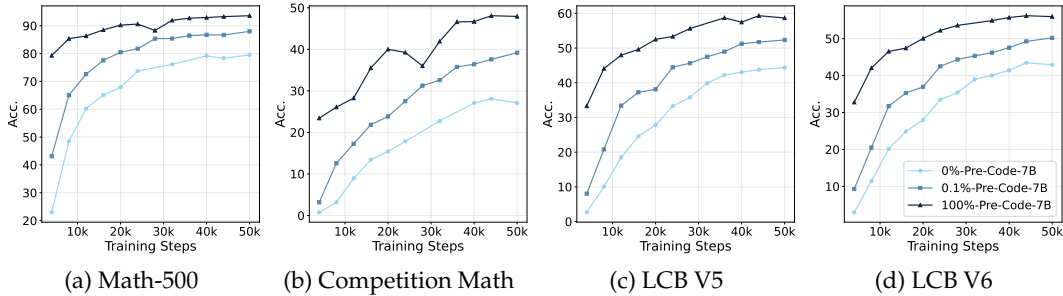


Figure 6: SFT evaluation curves using different amounts of pre-training backbones (7B).

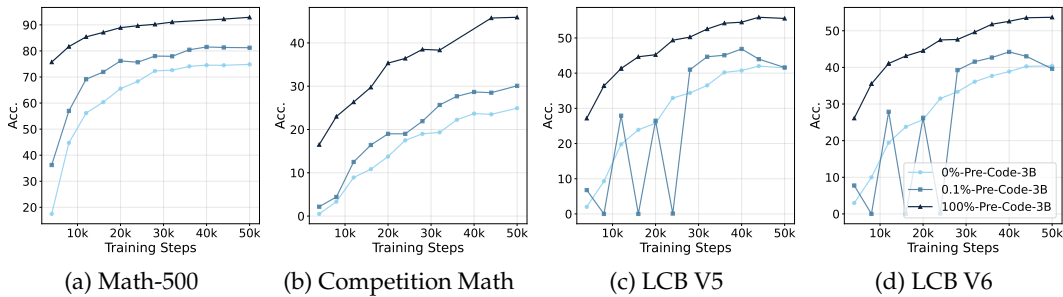


Figure 7: SFT evaluation curves using different amounts of pre-training backbones (3B).

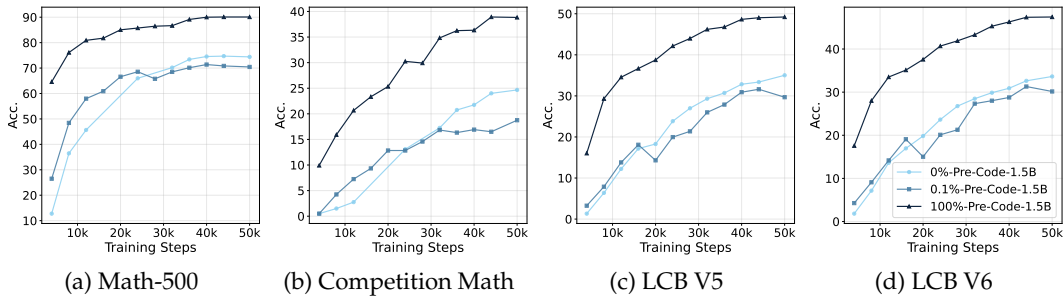


Figure 8: SFT evaluation curves using different amounts of pre-training backbones (1.5B).

## F Performance During Training of Competition Math Benchmarks

We present the performance curves of the 14/7/3/1.5B models for the competition math benchmarks in Figure 9, Figure 10, Figure 11 and Figure 12, respectively.

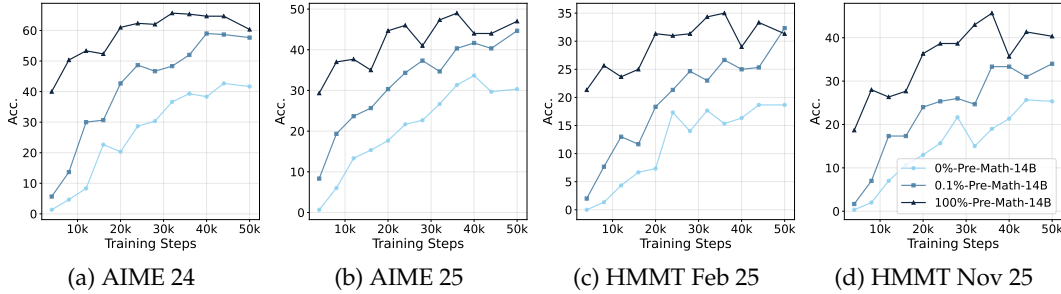


Figure 9: SFT evaluation curves over the math competition benchmarks, using different amounts of pre-training backbones (14B).

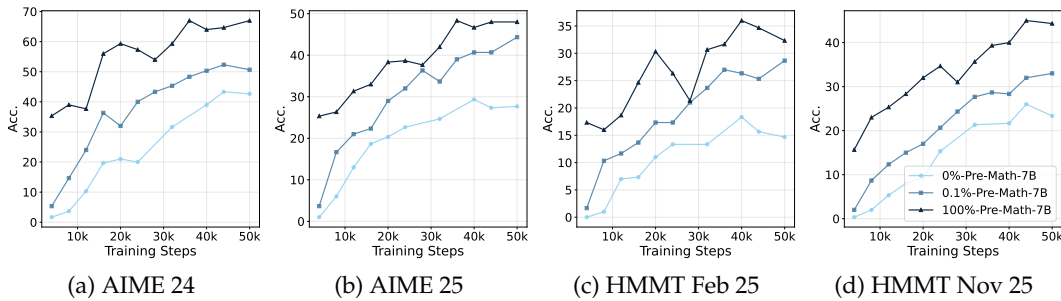


Figure 10: SFT evaluation curves over the math competition benchmarks, using different amounts of pre-training backbones (7B).

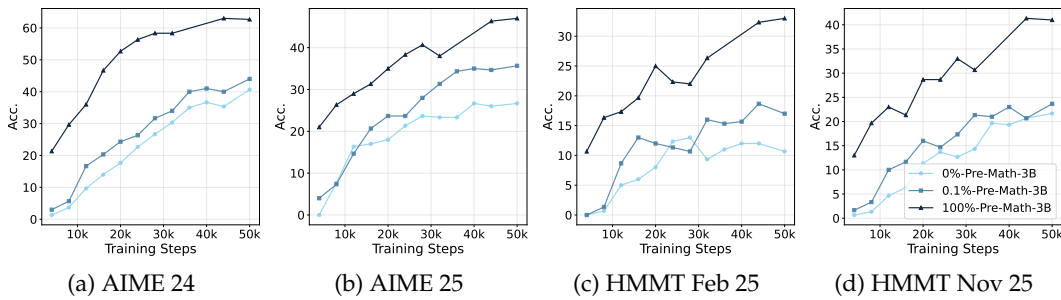


Figure 11: SFT evaluation curves over the math competition benchmarks, using different amounts of pre-training backbones (3B).

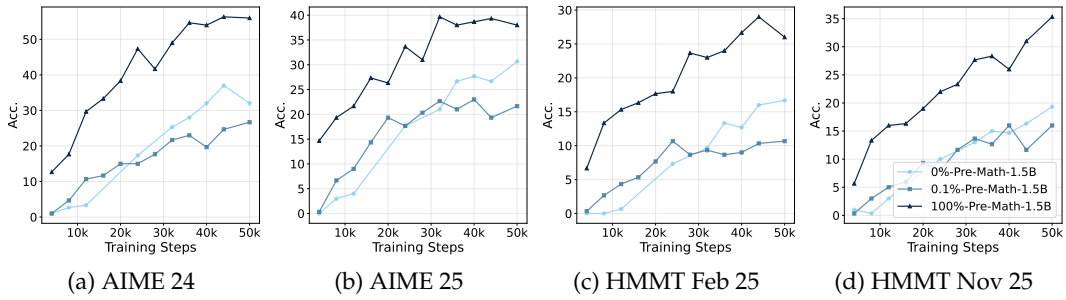


Figure 12: SFT evaluation curves over the math competition benchmarks, using different amounts of pre-training backbones (1.5B).