From Proof to Program: Characterizing Tool-Induced Reasoning Hallucinations in Large Language Models

Farima Fatahi Bayat, Pouya Pezeshkpour, Estevam Hruschka
Megagon Labs
{farima, pouya, estevam}@megagon.ai

Abstract

Tool-augmented Language Models (TaLMs) can invoke external tools to solve problems beyond their parametric capacity. However, it remains unclear whether these tool-enabled gains reflect trustworthy reasoning. Focusing on the Code Interpreter tool, we show that even when tools are selected and executed correctly, TaLMs treat tool outputs as substitutes for reasoning, producing solutions that appear correct but lack coherent justification. We term this failure mode **T**ool-**I**nduced **M**yopia (**TIM**), and study it using PyMath, a benchmark of 1,679 competition-level mathematical problems for which Python code is *helpful but not sufficient*. We further develop a multi-dimensional evaluation suite to quantify reasoning degradation in TaLMs relative to their non-tool counterparts. Our findings reveal that while TaLMs achieve up to a 19.3 *percentage point* gain in final-answer accuracy, their reasoning behavior consistently deteriorates (e.g., non-tool language models win up to 41.5% more often in pairwise comparisons of reasoning processes). This degradation intensifies with tool use; the more frequently a model invokes tools, the less coherent its reasoning becomes. Moreover, tool use shifts errors from arithmetic mistakes toward global reasoning failures (logic, assumption, creativity). Finally, we propose a preference-optimization-based framework that realigns TaLMs to use tool outputs as assistive evidence, improving both final-answer accuracy and the reasoning depth under tool use.[1]

1 Introduction

Large Language Models (LLMs) have grown increasingly capable, yet relying solely on their parametric knowledge introduces key limitations, including the inability to access real-time or domain-specific information (Yu and Ji, 2024; Wang et al., 2025), perform precise computations (Lu et al., 2023b), or fully comprehend user intentions (Qian et al., 2024). To address these shortcomings, Tool-augmented Reasoning (TIR) (Schick et al., 2023; Gou et al., 2024) has emerged as a promising paradigm. It enables LLMs to integrate natural language reasoning with external tools. Frontier LLMs (OpenAI, 2025b; GoogleAI, 2025c; Anthropic, 2025a) now offer native, sandboxed execution for select tools, choosing when to call, executing, and integrating results.

While tool calling significantly extends LLMs' utility in computation, retrieval, and procedural tasks, it also introduces new sources of failures. The most fundamental errors arise from *tool hallucinations*, where models either select inappropriate tools or misuse them, leading to incorrect or irrelevant outputs (Patil et al., 2025; Xu et al., 2025). However, even with correct tool selection and successful execution, LLMs can still generate non-factual outputs or flawed reasoning. Prior work ties factual errors to conflicts between parametric and retrieved knowledge (Sun et al., 2025) as well as to the propagation of errors from retrieved content (Magesh et al., 2024). Additional studies show that LLMs' unrestricted access to external tools can induce tool overuse (Qian et al., 2025), which confuses the model and harms performance. It also encourages cognitive offloading (Wang et al., 2025), which limits the model's use of its internal reasoning capabilities. Despite these observations, it remains unclear whether tool-augmented reasoning is hallucination-free even when other failure modes are controlled.

In this work, we focus exclusively on the Code Interpreter tool for mathematical problem solving to ensure strictly controlled tool use, with correct invocation and error-free execution. Our design avoids confounds introduced by multi-tool settings, such as API failures, interface mismatches, retrieval drift, or external data quality issues (Zhong et al., 2025; Faghih et al., 2025; Maekawa et al.,

[1] Code and data are available at: `https://github.com/megagonlabs/TIM`.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 42151–42169

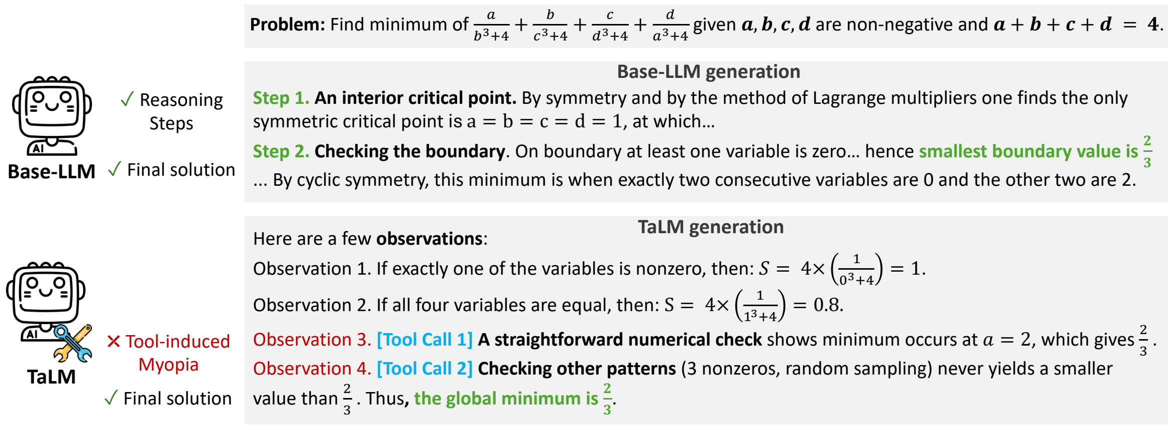

Figure 1: Comparison of Base LLM and Tool-augmented LLM (TaLM) reasoning. The Base LLM (top) derives the solution through step-by-step mathematical reasoning, while the TaLM (bottom) relies on **empirical checks** and multiple tool calls to search for the minimum, a failure mode characteristic of Tool-Induced Myopia (TIM).

2025; Han et al., 2025), thereby enabling a controlled investigation. Despite these idealized conditions, we identify a new class of hallucinations, which we term **T**ool-**I**nduced **M**yopia (TIM): a failure mode where access to an external tool (e.g., a Code Interpreter) causes the model to narrow its reasoning to what the tool can compute, rather than leveraging its full internal reasoning capabilities. TIM erodes user trust, as correct final answers can conceal flawed, tool-dependent reasoning.

Figure 1 illustrates this behavior: with Code Interpreter access (TaLM response), the model repeatedly performs **empirical checks** instead of producing the required reasoning steps generated by the same model without tools (Base-LLM response). Although the tool is correctly invoked and returns valid outputs, the model's reasoning depth is reduced. Importantly, existing evaluation approaches fail to capture this failure mode. Final-answer accuracy cannot detect it, since both responses are correct, and step-level logical-consistency metrics (Lightman et al., 2023; Zheng et al., 2025; Xia et al., 2025; Liu and Fang, 2025) also fail because the reasoning remains superficially coherent despite skipping essential logical steps. As a result, exposing TIM requires richer, multi-dimensional evaluation of how models reason under tool use.

To investigate TIM, we introduce PyMath, a dataset comprising 1,679 competition-level mathematical problems collected from multiple sources and curated to elicit, measure, and mitigate TIM under Code Interpreter access. We specifically target problems for which code-based computation is helpful but not sufficient for a complete solution (details in Section 3.1). We then quantify reasoning degradation in TaLMs compared to their Base (non-tool) counterparts using a comprehensive suite of reference-free and reference-based metrics (Section 3.2). Our results show that, even when TaLMs produce correct final answers (up to a 19.3 *percentage point* in final performance gains), they often overrely on tool outputs and produce shallower reasoning traces than non-tool models. This degradation intensifies with tool use: as models invoke tools more frequently, their reasoning becomes less coherent. The associated errors also shift from arithmetic mistakes to global reasoning failures involving logic, assumption, and creativity. Finally, our manual audit finds evidence of TIM in ~55% of high-risk model-generated solutions.

To mitigate this issue, we propose two complementary strategies: (1) a prompting intervention that encourages models to treat tools as reasoning aids, and (2) a DPO-based (Rafailov et al., 2023) preference-optimization that aligns TaLMs to integrate the Code Interpreter as an assistant that supports, rather than replaces, mathematical reasoning. On the evaluation split of PyMath, the fine-tuned TaLM demonstrates improved reasoning behavior and surpasses both the vanilla TaLM (+0.6%) and the Base LLM (+3.0%) in final-answer accuracy. In summary, our key contributions are:

- **Tool-Induced Myopia**: We identify TIM, a new class of TaLM hallucinations where tool access reduces reasoning depth even under idealized tool-use conditions.
- **Benchmark and Evaluation**: We introduce PyMath, a competition-level math benchmark with open-ended solutions, along with a multi-dimensional evaluation suite that reveals when and why TIM emerges and guides

safer tool integration.

- **Mitigation**: We propose a training-free prompting method and a preference-optimization fine-tuning approach that reduce TIM and improve reasoning depth in TaLMs.

2 Tool-induced Myopia

We define **Tool-Induced Myopia (TIM)** as a type of hallucination that arises in a controlled tool-use setting where the task requires substantial reasoning beyond what the tool alone can provide. In particular, we study competition-style mathematical problem solving with access to a Code Interpreter under two assumptions: (i) tool invocation is correct, and (ii) execution is error-free. In practice, it substitutes enumeration for proof, skips necessary derivations, mistakes empirical checks for universal guarantees (e.g., brute-force search), and may prematurely stop once code returns a plausible output. Note that using code solely for precise computation (e.g., evaluating a determinant or numerically finding a root), while also providing the necessary derivations, is **not** considered TIM. Figure 1 illustrates TIM: the Base-LLM solves the problem through step-by-step mathematical reasoning, whereas the TaLM shifts to an exploration-based approach, using empirical observations from internal computation and Code Interpreter to reach the correct answer through numerical search. TIM degrades reasoning by overrelying on tool outputs in place of the reasoning needed to justify the full solution. Notably, the TaLM in Figure 1 still produces the correct final answer via a logically coherent yet incomplete sequence of steps. To expose TIM, we next introduce an evaluation benchmark and a suite of targeted metrics.

3 Evaluating TIM in Language Models

3.1 PyMath

We focus on the domain of mathematical problem solving, where reasoning and computation are tightly coupled, and evaluate LLMs under two settings: with and without access to a Python Code Interpreter. We collect English, text-only competition-level math problems from multiple sources, summarized in Table 1. To mitigate data contamination, we restrict AIME (AIM, 2024-2025) to 2024–2025 problems. From Omni-Math (Gao et al., 2024), which provides difficulty ratings (1-10 scale), we retain only problems with difficulty ≥ 5, as frontier LLMs have already saturated performance on easier problems (OpenAI, 2025a; Anthropic, 2025b). Finally, we use the full set of problems from OlympiadBench (He et al., 2024) and OlympicArena (Huang et al., 2024).

Source	Total	U but not S	Eval (%)
AIME*	60	13	23.1
OlympiadBench	674	171	52.0
OlympicArena	169	47	12.8
Omni-Math	2569	1448	62.3
PyMath	-	**1679**	**59.5**

Table 1: Sources and statistics of competition-level math problems included in the PyMath benchmark (containing problems where Python is Useful but not Sufficient). *Only AIME 2024–2025 problems are included.

To elicit TIM in TaLMs more effectively, we target problems for which Python code is *helpful but not sufficient*. This setting creates a natural opportunity for tool use, while still *requiring* reasoning over tool outputs to derive a solution. To identify such cases, we adopt an LLM-as-a-judge protocol (Gu et al., 2025) to assess each problem along two dimensions: **(1) Python Usefulness**: whether Python code helps solve the problem; and **(2) Python Sufficiency**: whether Python code alone, without additional LLM reasoning, is sufficient to fully solve the problem. Appendix A.1 provides the filtering prompt and a manual validation of the judge.

Table 1 summarizes the resulting statistics by source. Our final dataset comprises 1,679 competition-level problems with step-by-step reference solutions: 1,000 problems (~60%, distributed across sources and problem difficulties) for evaluation, and the rest for TaLM training (10% development, 90% DPO fine-tuning; see Section 6).

3.2 Evaluation Suite

Evaluating LLMs' mathematical reasoning has traditionally relied on outcome-based evaluation, most predominantly final-answer accuracy (Liu et al., 2025a; Ahn et al., 2024). However, recent studies (Mondorf and Plank, 2024; Yee et al., 2024) have shown that LLMs can reach a correct final answer despite flawed reasoning. This gap has motivated a shift toward process-based evaluation, which assesses LLMs' *reasoning behavior* rather than their task performance.

Process-based approaches span two major families. *Reference-free* methods do not rely on gold reasoning traces and instead assess reasoning behavior via mechanisms such as self-consistency (Liu and Fang, 2025), or pairwise comparison (e.g., win rate (Chen et al., 2025; Qin et al., 2024b)). In contrast, *reference-dependent* approaches evaluate reasoning against a gold step-by-step solution, enabling fine-grained detection of invalid, missing, or inconsistent steps (Yan et al., 2025; Chernyshev et al., 2025). A recent line of work extends this idea via process reward models (PRMs) to score partial solutions step by step, providing a learned measure of reasoning soundness without requiring gold supervision (Lightman et al., 2023; Zheng et al., 2025; Li et al., 2025).

As illustrated in Figure 1, neither final-answer accuracy nor unidimensional process-based metrics (e.g., PRM scores alone) are sufficient to expose TIM in TaLMs: the model may output the correct answer and generate seemingly coherent reasoning steps while still exhibiting degraded reasoning under tool use. This motivates the need for a multi-dimensional evaluation suite that jointly captures: **(i)** task outcome, **(ii)** counterfactual impact of tool use, **(iii)** divergence from ground truth reasoning traces, and **(iv)** step-level logical consistency. Next, we introduce these four evaluation dimensions and clarify why each one is necessary, yet insufficient on its own, for diagnosing TIM.

Final-Answer Accuracy: We first measure final-answer accuracy as the standard measure of task success. Although this metric cannot reveal whether a TaLM solved the problem through mathematical reasoning or tool-driven shortcuts, it confirms that TIM failure affects reasoning, not task performance. For all subsequent metrics which target reasoning behavior, we evaluate models only on problems with correct final answer to isolate the effect of tool use on how the model reasons, rather than whether it succeeds.

Win Rate: To assess whether tool use meaningfully affects reasoning behavior, we compare solutions from a TaLM to those of the corresponding Base (non-tool) LLM. Following prior work showing that LLMs can reliably approximate human preferences (Zheng et al., 2023; Chen et al., 2025), we use an LLM judge with a rubric adapted from Petrov et al. (2025) covering four error types: (1) *logic errors* (logical fallacies or unjustified leaps), (2) *assumption errors* (unsupported or incorrect assumptions), (3) *creativity errors* (invalid solution strategies), and (4) *algebra/arithmetic errors* (critical symbolic or numeric mistakes). To avoid order bias, we randomize the presentation order of TaLM and Base-LLM solutions (Shi et al., 2025). The judge then selects the solution with fewer errors as the winner (prompt in Appendix A.2).

Win Rate measures the proportion of comparisons in which the TaLM solution is preferred (and vice versa). A lower Win Rate indicates tool access *harms* the LLM's reasoning behavior. This comparison-based setup reduces single-judge bias (Liu et al., 2025b; Fatahi Bayat et al., 2025) and captures the counterfactual effect of tool use.

Miss Rate: Inspired by recall-based measures in long-form factuality evaluation (Wei et al., 2024; Liu et al., 2025b), we define *Miss Rate* as the proportion of reasoning steps in the ground-truth solution that are absent from the model-generated solution, i.e., $\frac{|\text{missing steps}|}{|\text{gold steps}|}$. A high Miss Rate indicates that the model abandoned a valid derivation, skipped necessary steps, or replaced reasoning with trial-and-error code execution. Unlike final-answer accuracy, Miss Rate detects invalid reasoning paths even when the model reaches the correct solution. However, this metric may over-penalize a solution when the reference solution(s) do not cover all valid reasoning paths (prompt in Appendix A.3).

PRM Accuracy: Finally, we evaluate step-level reasoning behavior using a Process Reward Model, specifically QWEN2.5-MATH-7B-PRM800K (Zheng et al., 2025), trained on $\sim$800K annotated mathematical steps. We aggregate step-level scores to obtain an overall correctness score for each solution. However, while PRMs evaluate step correctness, they may fail to detect reasoning shortcuts where code outputs replace derivations, as they lack a holistic view of the solution. In our analysis, we use PRM scores primarily to compare Base-LLM and TaLM solutions, and Appendix A.4 shows that this comparative signal remains stable under moderate perturbations.

These four metrics jointly provide minimal yet complete coverage of TIM: Final-answer accuracy captures outcomes; Win Rate isolates the tool's counterfactual impact on reasoning behavior; Miss Rate quantifies divergence from valid reference reasoning; and a PRM assesses step-level soundness without relying on the ground truth. This set is sufficient because TIM exhibits a clear and convergent pattern: final-answer accuracy stays the same or improves in TaLMs, while Win Rate declines, Miss Rate increases, and PRM accuracy drops. This suite both distinguishes TIM from alternative explanations (e.g., general model weakness or genuinely helpful tool use) and is falsifiable: when tools are removed, the pattern recedes, and when tool reliance is increased, it strengthens. Together, these properties can establish both the existence and mechanism of Tool-Induced Myopia.

4 Experimental Setup

Large Language Models: We evaluate frontier proprietary language models equipped with in-house Code Interpreters. These models can autonomously decide when to invoke tools, allowing us to study tool-augmented reasoning without modifying or intervening in their internal execution pipeline. We benchmark seven models across three families: (1) OpenAI GPT-4.1-mini and GPT-4.1 (non-thinking LLMs; OpenAI, 2025c), and o4-mini (OpenAI, 2025d) and GPT-5 (OpenAI, 2025a) (thinking LLMs), (2) Gemini-2.0-Flash (non-thinking; GoogleAI, 2025a) and Gemini-2.5-Flash (thinking; GoogleAI, 2025b), (3) Claude-Opus-4 (thinking; Anthropic, 2025b).

Evaluation Benchmark: We use the evaluation split of PyMath as our benchmark dataset and apply our four-dimensional evaluation suite to measure reasoning and tool-use behavior. We use GPT-5 as the judge for all LLM-as-a-judge evaluations.

5 Results and Analyses

In this section, we first evaluate Base-LLMs and their tool-augmented counterparts (TaLMs) using our four-dimensional evaluation suite (Section 5.1). We then examine how the severity of TIM varies with the number of tool calls (Section 5.2) and analyze whether the complexity of TaLM-generated code correlates with reasoning hallucinations (Appendix A.5). Next, we apply the error taxonomy from Section 3.2 to characterize error types introduced by tool access (Section 5.3), and analyze tool invocation frequency given that PyMath encourages, but does not guarantee, tool use (Section 5.4). Finally, Section 5.5 presents a qualitative analysis confirming that solutions flagged as high risk by our metrics indeed exhibit TIM behavior.

5.1 Base-LLMs Show Stronger Reasoning Despite Lower Accuracy

Our benchmarking results on the evaluation set of PyMath are presented in Table 2. We report Final-answer Accuracy, Miss Rate, Win Rate, and PRM Accuracy across the Base and TaLM variants of seven LLMs. The results reveal a consistent pattern characteristic of TIM: although **TaLMs achieve higher Final-answer Accuracy, Base-LLMs exhibit stronger mathematical reasoning**, reflected in lower Miss Rates, higher Win Rates, and higher PRM Accuracy on average. Prior work has shown that LLMs can produce the correct final answer despite flawed or incomplete reasoning (Mondorf and Plank, 2024). Our findings demonstrate that access to external tools often amplifies this discrepancy.

Interestingly, the highest PRM scores are achieved by non-thinking GPT-4.1 models compared to their stronger "thinking" variants. This is inline with recent findings that step-level reward models struggle to reliably assess long and complex reasoning chains, often conflating fluency with correctness, becoming miscalibrated on stronger models, and generalizing poorly to the longer, self-correcting traces produced by advanced reasoning models (Bamba et al., 2025; Lee et al., 2025).

Finally, we posit that the reasoning gap between Base-LLM and TaLM appears moderate in aggregate because TIM is partially masked by three factors: (1) our filtering for TIM-prone problems, while targeted, is not perfect; (2) existing metrics have limited sensitivity to TIM hallucinations; and (3) tool use yields only modest accuracy gains for most LLMs. For subsequent analyses, we focus only on solutions with correct final answers.

5.2 Base-TaLM Reasoning Gap Widens with Number of Tool Calls

We examine how reasoning behavior varies with tool-call frequency, a key factor underlying TIM: as tool reliance grows, models increasingly substitute computation for reasoning. We test whether higher number of calls amplify the Base–TaLM gap when final answers are correct. We group problems into bins based on the number of tool calls in their TaLM-generated solutions: {0–3, 4–7, 8–11, 12+}. Figure 2 shows three consistent trends across most models. First, the Base model's *Win Rate* against the TaLM increases with tool-call frequency, indicating degraded reasoning under heavier tool reliance. Second, the *Miss Rate* generally rises with increased tool calls, reflecting greater divergence from reference reasoning. Third, *PRM Accuracy* typically declines as tool calls increase, suggesting that longer, tool-heavy trajectories accumulate step-level errors. Occasional reversals in the 12+ bin are due to limited sample sizes. We further verify

Model	Variant	Final Acc. (↑)	Miss Rate (↓)	Win Rate (↑)	PRM Acc. (↑)
GPT-4.1-mini	Base	30.0	45.7	58.6	93.0
	TaLM	28.7	49.9	41.4	88.9
GPT-4.1	Base	24.6	48.1	54.4	88.6
	TaLM	27.0	49.9	45.6	85.9
o4-mini	Base	45.1	45.5	49.0	73.6
	TaLM	64.4	47.6	50.9	67.9
GPT-5-Thinking	Base	67.5	38.8	56.0	57.5
	TaLM	71.9	43.8	44.0	50.2
Gemini-2.0-Flash	Base	24.3	54.2	52.7	65.0
	TaLM	25.1	56.6	47.3	68.5
Gemini-2.5-Flash	Base	45.4	40.2	54.6	81.5
	TaLM	45.7	40.9	45.4	78.8
Claude-Opus-4	Base	28.0	50.9	41.3	77.9
	TaLM	40.5	52.8	58.6	57.8
Average	Base	37.8	**46.2**	**52.4**	**76.7**
	TaLM	**43.3**	48.8	47.6	71.1

Table 2: Performance of Base-LLMs and TaLMs across four evaluation metrics. Best score in each column is highlighted in green. On average, TaLMs achieve higher Final-answer Accuracy (Final Acc.), yet Base-LLMs exhibit greater reasoning depth, confirming the presence of TIM.

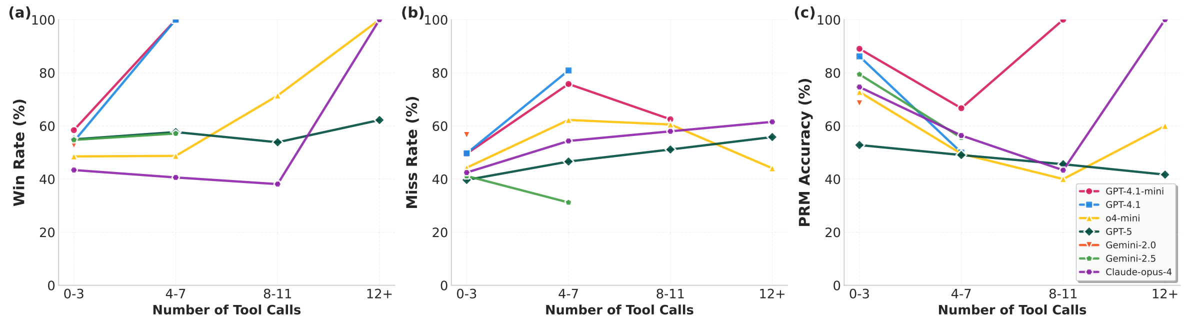

Figure 2: Impact of tool-call frequency on TaLM reasoning. Across tool-call bins (0–3 to 12+), higher tool-call frequency is associated with (a) increased Base-over-TaLM Win Rate, (b) higher Miss Rate, and (c) lower PRM Accuracy, indicating a widening reasoning gap as reliance on tools grows.

that this effect is not explained by the syntactic or structural complexity of generated code; detailed analysis shows no correlation between code complexity and TIM severity (Appendix A.5).

5.3 Tool Use Induces Global Reasoning Errors

In this section, we analyze how reasoning error types shift under tool use by comparing TaLM solutions with those of Base-LLMs. Using the error taxonomy adapted from Petrov et al. (2025), we prompt our LLM judge (GPT-5) to rate each error type on a 1–5 scale, or output None if no error is detected (prompt in Appendix A.6).

Across models, tool use induces a consistent shift in error patterns (Figure 3). Logic, assumption, and creativity errors increase in nearly all cases, indicating more unjustified inferences and reasoning shortcuts under tool reliance. In contrast, algebraic and arithmetic errors decrease as computation is delegated to the Code Interpreter, while the rate of error-free solutions also drops. These results show that TIM is not merely about missing steps, but reflects a fundamental change in how models reason under tool use.

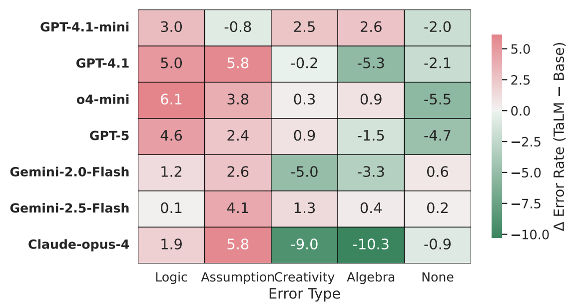

Figure 3: Change in reasoning error rates after tool use (Δ = TaLM − Base). Positive values indicate higher error frequency with Code Interpreter access.

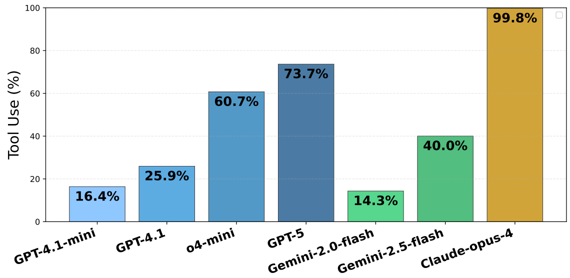

Figure 4: Tool use rate across TaLMs. Thinking TaLMs (GPT-5, o4-mini, Gemini-2.5-Flash) invoke code on ~50% more problems than non-thinking models.

5.4 Thinking Models Use Code More Often

Having shown that more tool calls intensify TIM, we now quantify how often each model invokes the Code Interpreter tool on math problems, providing a "dose" measure of TIM risk. We investigate tool use rates across TaLMs on PyMath, which was curated to encourage code use. Figure 4 shows the percentage of problems on which each TaLM invoked the Code Interpreter tool. Thinking models exhibit substantially higher tool usage than non-thinking models: Claude-opus-4 achieves the highest rate at 99.8%, followed by GPT-5 at 73.7%, while non-thinking models show more modest usage. **On average, thinking models invoke code on 49.7% more problems.** Given the established relationship between tool-call frequency and TIM severity (Section 5.2), this heavier reliance suggests that thinking models face higher TIM risk.

5.5 TIM in Over Half of High-Risk Solutions

We conduct a targeted manual evaluation to validate TIM qualitatively and identify recurring linguistic cues that precede it. We focus on high-risk TaLM solutions that satisfy three criteria: (1) a correct final answer, (2) Base-LLM solution is preferred over the TaLM solution in Win Rate judgment, and (3) PRM flags errors in the TaLM's reasoning trace. From this filtered set, we select the top 10 samples per model ranked by Miss Rate to assess TIM presence and characteristic precursor phrases.

Across models, **TIM appears in 54.3% of high-risk cases**, demonstrating strong alignment between our automated metrics. Incidence is higher for models with greater tool use (Figure 4), such as Claude-Opus-4 and o4-mini, and lower for Gemini models. Less capable models often display explicit precursor cues (e.g., "one numerically finds," "systematic checks show," or "let's verify programmatically."), whereas stronger models like GPT-5 silently substitute code outputs for derivations (more details and a list of precursor phrases in Appendix A.7). These cues are indicative rather than definitive and should be interpreted alongside quantitative metrics.

6 Mitigating TIM Hallucination

Our evaluation results reveal widespread TIM hallucinations, where models substitute tool outputs for mathematical reasoning. In this section, we propose two complementary mitigation strategies that encourage models to use the Code Interpreter as an *assistant to reasoning* rather than a substitute.

6.1 Prompting-Based Mitigation

We design a lightweight prompting strategy that self-instructs the model to treat tool outputs as reasoning aids. Specifically, after each problem statement, we inject the following instruction:

> *"We should treat code snippets and their execution results only as helpful hints, and derive the solution through mathematical reasoning."*

This single-sentence intervention promotes mathematical thinking over computational shortcuts.

6.2 Alignment-Based Mitigation

We view TIM primarily as a behavioral misalignment issue: under tool use, the model tends to treat Code Interpreter as a search mechanism that substitutes for derivation, rather than as an assistant that supports reasoning. This suggests that mitigation should target *how* the model reasons with tool outputs, not merely whether it reaches the correct answer. We therefore adopt Direct Preference Optimization (DPO; (Rafailov et al., 2023)), which is well-suited for aligning relative preferences between alternative reasoning trajectories under the same problem and tool context. Due to resource constraints, we focus on fine-tuning a single model (GPT-4.1) to test this approach.

6.2.1 Preference Data Creation

We construct a preference dataset from the training split of PyMath. Because TIM is a reasoning failure rather than an answer-correctness failure, we explicitly control for correctness in our preference construction: both chosen and rejected samples are required to preserve the same intermediate and final results. Thus, the preference signal cannot be

Variant	**Final Acc. (↑)**	**Miss Rate (↓)**	**Win Rate vs. TaLM (↑)**	**PRM Acc. (↑)**
Base	24.6	48.1	54.4	**88.6**
TaLM	27.0	49.9	–	85.9
TaLM + Prompting	25.1	49.4	52.7	82.9
TaLM + DPO	**27.6**	**46.6**	**58.2**	83.3

Table 3: Impact of mitigation strategies on tool-augmented GPT-4.1. Prompting reduces TIM without training but at the cost of accuracy, while DPO improves both accuracy and reasoning quality.

explained by answer correctness, but instead isolates differences in reasoning behavior under tool use. Accordingly, for each problem, we construct a *chosen-rejected* pair, where the chosen solution exhibits stronger reasoning under tool use and the rejected solution exhibits TIM characteristics.

Chosen samples: We generate chosen solutions by applying the prompting strategy from Section 6.1 to GPT-4.1, which encourages mathematical reasoning under tool use.

Rejected samples: We generate rejected examples through controlled degradation: given a problem and its corresponding chosen solution, we prompt the same model to rewrite a solution span with excessive reliance on tool outputs, producing a coherent but tool-dependent solution that omits or abbreviates intermediate reasoning steps. This process creates naturalistic rejected samples with TIM hallucination (more details in Appendix A.8).

6.3 Results

Table 3 presents mitigation results on our evaluation benchmark. Both mitigation strategies reduce TIM hallucination compared to the vanilla TaLM, but with distinct trade-offs.

Prompting intervention. Zero-shot prompting substantially improves reasoning without model retraining, improving Win Rate from 45.6% to 52.7% and slightly reducing Miss Rate (49.9% to 49.4%). This improvement comes at a cost to final-answer accuracy, which drops from 27.0% to 25.1%, approaching the Base-LLM's 24.6%.

DPO alignment. The fine-tuned model achieves the strongest overall trade-off: it attains the highest final-answer accuracy (27.6%, +0.6 points over the vanilla TaLM), the lowest Miss Rate (46.6%), and is preferred over the vanilla TaLM in 58.2% of pairwise comparisons. However, its PRM accuracy (83.3%) remains below that of the Base-LLM (88.6%), suggesting that while DPO mitigates global reasoning failures, it does not fully recover step-level correctness. The largest gains appear on the reasoning-sensitive metrics, especially Miss Rate and Win Rate, while final-answer accuracy improves only modestly. These results suggest that DPO primarily acts as a behavioral alignment mechanism rather than a capability boost.

DPO alignment. The fine-tuned model achieves the strongest overall performance: highest final-answer accuracy (27.6%, +0.6% over vanilla TaLM), highest Win Rate (58.2% vs. 54.4% for the Base-LLM), and lowest Miss Rate (46.6%). However, PRM accuracy (83.3%) remains below that of the Base-LLM (88.6%), suggesting that while DPO mitigates global reasoning failures, it does not fully restore step-level correctness. Notably, the largest gains appear in the reasoning-sensitive metrics, especially Win Rate and Miss Rate, while final-answer accuracy improves only modestly. This pattern is consistent with a behavioral alignment effect: DPO primarily changes how the model uses tool outputs during reasoning, rather than simply improving task capability.

7 Related Work

Tool-augmented reasoning extends LLMs with external programmatic interfaces to overcome limitations of parametric knowledge. Early work, such as Toolformer (Schick et al., 2023), showed that models can learn when and how to invoke tools, while later surveys (Wang et al., 2024; Gou et al., 2024) define LLM-used tools and provide broad overviews of tool-augmented LMs. This has led to the development of diverse tools, such as web search (Asai et al., 2024; Lu et al., 2023a), code interpreters (Chen et al., 2023; Gao et al., 2023a), and domain-specific APIs (Qin et al., 2023).

Hallucinations in TaLMs. Despite these advances, TaLMs exhibit persistent failure modes. Prior work has studied *tool hallucinations*, where models misuse or select incorrect tools (Xu et al., 2025), as well as factual errors arising from conflicts between parametric and retrieved knowledge (Sun et al., 2025) or from inaccurate retrieved content (Magesh et al., 2024). Other studies highlight attribution failures in retrieval-augmented generation (Gao et al., 2023b; Liu et al., 2023) and cognitive offloading, where unrestricted tool access suppresses internal reasoning (Qian et al., 2025; Wang et al., 2025). In contrast, we show that *even when tools are correctly invoked and executed*, TaLMs exhibit Tool-Induced Myopia, which causes them to substitute reasoning with computation.

Evaluating Mathematical Reasoning. Evaluation of mathematical reasoning has traditionally relied on final-answer accuracy using benchmarks such as GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). However, outcome-based metrics fail to capture reasoning behavior, especially as frontier models approach saturation (Mondorf and Plank, 2024). Recent work has proposed process-level evaluation (Lightman et al., 2023; Zheng et al., 2025), chain-of-thought scoring (Xia et al., 2025; Li et al., 2025), and model-based judges (Zhou et al., 2025; Xu et al., 2025). However, even with competition-level benchmarks (AIM, 2024-2025; He et al., 2024; Gao et al., 2024), existing evaluations remain largely one-dimensional and fail to distinguish robust reasoning from tool-driven shortcuts. We address this gap with our benchmark and multi-dimensional evaluation suite explicitly designed to surface TIM.

Mitigating TaLM Failures. Prior mitigation strategies focus on prompting-based self-correction (Shinn et al., 2023), alignment techniques targeting tool selection and invocation errors (Qin et al., 2024a; Patil et al., 2024; Ross et al., 2025; Xu et al., 2025), and hybrid approaches that combine parametric and non-parametric reasoning (Asai et al., 2024). While effective for tool misuse, these approaches do not address reasoning degradation under correct tool use. Our work introduces interventions that explicitly reward comprehensive analytical reasoning over premature tool reliance, providing the first systematic approach to mitigating TIM.

8 Conclusion

Investigating tool-augmented reasoning across competition-level math, we show that even with correct tool use, TaLMs exhibit **Tool-Induced Myopia (TIM)**, substituting computation for mathematical reasoning. To study this, we introduced PYMATH and a four-dimensional evaluation suite that reveals a consistent pattern across seven frontier models: TaLMs improve final-answer accuracy but produce weaker reasoning than Base-LLMs. Error analysis shows fewer arithmetic errors but more logical and assumption errors, especially with increased tool use. Finally, we introduced two mitigation strategies: prompting and DPO-based preference optimization, which reduce TIM hallucination and improve trustworthy reasoning.

9 Limitations

While our study provides the first characterization of Tool-Induced Myopia (TIM), it also has several limitations: First, to ensure precise control over tool invocation and execution, we restrict our analysis to a single tool, the Code Interpreter. This choice eliminates confounds such as API failures or retrieval noise, but it limits the generality of our findings to other tool types (e.g., search, retrieval, or various APIs). Future work should extend this framework to a broader range of tools and interaction settings. Second, our manual investigation of TIM occurrence and linguistic precursors is limited in scope. While qualitative inspection confirms strong alignment with our automated metrics, a larger-scale human evaluation, covering more models, domains, and annotators, would strengthen the robustness and generalizability of these observations. Finally, due to computational and resource constraints, our mitigation experiments focus exclusively on the GPT-4.1. Although this choice provides a representative case for high-end tool-augmented reasoning, reproducing the mitigation analyses on additional LLMs remains an important direction for validating consistency and scalability.

10 Ethics Statement

This work studies the reasoning behavior of tool-augmented large language models using publicly available mathematical problem datasets and model-generated outputs. No human subjects were involved in data collection, and all qualitative annotations were performed by the authors on model outputs only.

We use proprietary language models accessed via official APIs and follow the providers' usage policies. Our analysis focuses on identifying and mitigating reasoning failures in tool-augmented models, with the goal of improving reliability and trustworthiness in deployed systems. We do not release any personally identifiable information, and all data used or released are either publicly available or derived from synthetic model outputs.

AI assistants, ChatGPT and Claude models, were used for writing support and minor editing. All scientific contributions, experimental design, analysis, and conclusions are the responsibility of the authors.

References

2024-2025. Aime i and aime ii problems and solutions. `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`.

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 225–237, St. Julian's, Malta. Association for Computational Linguistics.

Anthropic. 2025a. Claude models. https://docs.claude.com/en/docs/about-claude/models/overview. Version: 2025-11-10.

Anthropic. 2025b. System card: Claude opus 4 and claude sonnet 4. Version: 2025-05.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. *Proceedings of the International Conference on Learning Representations (ICLR) 2024*.

Udbhav Bamba, Heng Yang, Rishabh Tiwari, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2025. Reward under attack: Evaluating the sensitivity of process reward models.

Barry W. Boehm. 1981. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ.

Liguo Chen, Qi Guo, Hongrui Jia, Zhengran Zeng, Xin Wang, Yijiang Xu, Jian Wu, Yidong Wang, Qing Gao, Jindong Wang, and 1 others. 2024. A survey on evaluating large language models in code generation tasks. *arXiv preprint arXiv:2408.16498*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Xilun Chen, Ilia Kulikov, Vincent-Pierre Berges, Barlas Oğuz, Rulin Shao, Gargi Ghosh, Jason Weston, and Wen tau Yih. 2025. Learning to reason for factuality. *Preprint*, arXiv:2508.05618.

Konstantin Chernyshev, Vitaliy Polshkov, Vlad Stepanov, Alex Myasnikov, Ekaterina Artemova, Alexei Miasnikov, and Sergei Tilga. 2025. U-math: A university-level benchmark for evaluating mathematical skills in large language models. In *Proceedings of the Fourth Workshop on Generation, Evaluation and Metrics (GEM^2)*, pages 974–1001.

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. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Shihan Dou, Haoxiang Jia, Shenxi Wu, Huiyuan Zheng, Weikang Zhou, Muling Wu, Mingxu Chai, Jessica Fan, Caishuang Huang, Yunbo Tao, and 1 others. 2024. What's wrong with your code generated by large language models? an extensive study. *arXiv preprint arXiv:2407.06153*.

Kazem Faghih, Wenxiao Wang, Yize Cheng, Siddhant Bharti, Gaurang Sriramanan, Sriram Balasubramanian, Parsa Hosseini, and Soheil Feizi. 2025. Tool preferences in agentic llms are unreliable. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20965–20980.

Farima Fatahi Bayat, Lechen Zhang, Sheza Munir, and Lu Wang. 2025. FactBench: A dynamic benchmark for in-the-wild language model factuality evaluation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33090–33110, Vienna, Austria. Association for Computational Linguistics.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. *Preprint*, arXiv:2410.07985.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023a. Pal: Program-aided language models. *Proceedings of the 40th International Conference on Machine Learning*.

Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. 2023b. Enabling large language models to generate text with citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6465–6488, Singapore. Association for Computational Linguistics.

GoogleAI. 2025a. Gemini 2.0: Flash, flash-lite and pro. https://developers.googleblog.com/en/gemini-2-family-expands/.

GoogleAI. 2025b. Gemini 2.5 flash best for fast performance on everyday tasks. https://deepmind.google/models/gemini/flash/.

GoogleAI. 2025c. Gemini models. https://ai.google.dev/gemini-api/docs/models. Version: 2025-11-10.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. 2025. A survey on llm-as-a-judge. *Preprint*, arXiv:2411.15594.

Ziwen Han, Meher Mankikar, Julian Michael, and Zifan Wang. 2025. Search-time data contamination. *arXiv preprint arXiv:2508.13180*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, Bangkok, Thailand. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1.

Zhen Huang, Zengzhi Wang, Shijie Xia, Xuefeng Li, Haoyang Zou, Ruijie Xu, Run-Ze Fan, Lyumanshan Ye, Ethan Chern, Yixin Ye, Yikai Zhang, Yuqing Yang, Ting Wu, Binjie Wang, Shichao Sun, Yang Xiao, Yiyuan Li, Fan Zhou, Steffi Chern, and 9 others. 2024. Olympicarena: Benchmarking multi-discipline cognitive reasoning for superintelligent ai. In *Advances in Neural Information Processing Systems*, volume 37, pages 19209–19253. Curran Associates, Inc.

Dong Bok Lee, Seanie Lee, Sangwoo Park, Minki Kang, Jinheon Baek, Dongki Kim, Dominik Wagner, Jiongdao Jin, Heejun Lee, Tobias Bocklet, Jinyu Wang, Jingjing Fu, Sung Ju Hwang, Jiang Bian, and Lei Song. 2025. Rethinking reward models for multi-domain test-time scaling. *Preprint*, arXiv:2510.00492.

Vladimir I. Levenshtein. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10(8).

Ruosen Li, Ziming Luo, and Xinya Du. 2025. FG-PRM: Fine-grained hallucination detection and mitigation in language model mathematical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 4247–4278, Suzhou, China. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

MingShan Liu and Jialing Fang. 2025. Enhancing mathematical reasoning in large language models with self-consistency-based hallucination detection. *Preprint*, arXiv:2504.09440.

Nelson Liu, Tianyi Zhang, and Percy Liang. 2023. Evaluating verifiability in generative search engines. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7001–7025, Singapore. Association for Computational Linguistics.

WenTao Liu, Hanglei Hu, Jie Zhou, Yuyang Ding, Junsong Li, Jiayi Zeng, MengLiang He, Qin Chen, Bo Jiang, Aimin Zhou, and Liang He. 2025a. Mathematical language models: A survey. *ACM Comput. Surv.* Just Accepted.

Xin Liu, Lechen Zhang, Sheza Munir, Yiyang Gu, and Lu Wang. 2025b. VeriFact: Enhancing long-form factuality evaluation with refined fact extraction and reference facts. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17919–17936, Suzhou, China. Association for Computational Linguistics.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023a. Chameleon: plug-and-play compositional reasoning with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. 2023b. A survey of deep learning for mathematical reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14605–14631, Toronto, Canada. Association for Computational Linguistics.

Seiji Maekawa, Jackson Hassell, Pouya Pezeshkpour, Tom Mitchell, and Estevam Hruschka. 2025. Towards reliable benchmarking: A contamination free, controllable evaluation framework for multi-step llm function calling. *arXiv preprint arXiv:2509.26553*.

Varun Magesh, Faiz Surani, Matthew Dahl, Mirac Suzgun, Christopher D. Manning, and Daniel E. Ho. 2024. Hallucination-free? assessing the reliability of leading ai legal research tools. *Preprint*, arXiv:2405.20362.

T.J. McCabe. 1976. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320.

Philipp Mondorf and Barbara Plank. 2024. Beyond accuracy: Evaluating the reasoning behavior of large language models – a survey. In *First Conference on Language Modeling*.

OpenAI. 2025a. Gpt-5 system card. https://cdn.openai.com/gpt-5-system-card.pdf. Version: 2025-08-13.

OpenAI. 2025b. Gpt models. https://platform.openai.com/docs/models. Version: 2025-11-10.

OpenAI. 2025c. Introducing gpt-4.1 in the api. https://openai.com/index/gpt-4-1/. Version: 2025-04-14.

OpenAI. 2025d. o4-mini. https://platform.openai.com/docs/models/o4-mini.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565. Curran Associates, Inc.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2025. Gorilla: large language model connected with massive apis. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Ivo Petrov, Jasper Dekoninck, Lyuben Baltadzhiev, Maria Drencheva, Kristian Minchev, Mislav Balunović, Nikola Jovanović, and Martin Vechev. 2025. Proof or bluff? evaluating llms on 2025 usa math olympiad. *Preprint*, arXiv:2503.21934.

Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi Chen, Avirup Sil, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. SMART: Self-aware agent for tool overuse mitigation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 4604–4621, Vienna, Austria. Association for Computational Linguistics.

Cheng Qian, Bingxiang He, Zhong Zhuang, Jia Deng, Yujia Qin, Xin Cong, Zhong Zhang, Jie Zhou, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024. Tell me more! towards implicit user intention understanding of language model driven agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1088–1113, Bangkok, Thailand. Association for Computational Linguistics.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, and 24 others. 2024a. Tool learning with foundation models. *ACM Comput. Surv.*, 57(4).

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ArXiv*, abs/2307.16789.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024b. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Representation Learning*, volume 2024, pages 9695–9717.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: your language model is secretly a reward model. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Hayley Ross, Ameya Sunil Mahabaleshwarkar, and Yoshi Suhara. 2025. When2Call: When (not) to call tools. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3391–3409, Albuquerque, New Mexico. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Lin Shi, Chiyu Ma, Wenhua Liang, Xingjian Diao, Weicheng Ma, and Soroush Vosoughi. 2025. Judging the judges: A systematic study of position bias in llm-as-a-judge. *Preprint*, arXiv:2406.07791.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Zhongxiang Sun, Xiaoxue Zang, Kai Zheng, Yang Song, Jun Xu, Xiao Zhang, Weijie Yu, Yang Song, and Han Li. 2025. Redeep: Detecting hallucination in retrieval-augmented generation via mechanistic interpretability. *Preprint*, arXiv:2410.11414.

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. 2025. Acting less is reasoning more! teaching model to act efficiently. *Preprint*, arXiv:2504.14870.

Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. 2024. What are tools anyway? a survey from the language model perspective. *Preprint*, arXiv:2403.15452.

Jerry Wei, Chengrun Yang, Xinying Song, Yifeng Lu, Nathan Hu, Jie Huang, Dustin Tran, Daiyi Peng, Ruibo Liu, Da Huang, Cosmo Du, and Quoc V. Le. 2024. Long-form factuality in large language models. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NeurIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Shijie Xia, Xuefeng Li, Yixin Liu, Tongshuang Wu, and Pengfei Liu. 2025. Evaluating mathematical reasoning beyond accuracy. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press.

Hongshen Xu, Zichen Zhu, Lei Pan, Zihan Wang, Su Zhu, Da Ma, Ruisheng Cao, Lu Chen, and Kai Yu. 2025. Reducing tool hallucination via reliability alignment.

Yuchen Yan, Jin Jiang, Zhenbang Ren, Yijun Li, Xudong Cai, Yang Liu, Xin Xu, Mengdi Zhang, Jian Shao, Yongliang Shen, and 1 others. 2025. Verifybench: Benchmarking reference-based reward systems for large language models. *arXiv preprint arXiv:2505.15801*.

Evelyn Yee, Alice Li, Chenyu Tang, Yeon Ho Jung, Ramamohan Paturi, and Leon Bergen. 2024. Faithful and unfaithful error recovery in chain of thought. In *First Conference on Language Modeling*.

Pengfei Yu and Heng Ji. 2024. Information association for language model updating by mitigating LM-logical discrepancy. In *Proceedings of the 28th Conference on Computational Natural Language Learning*, pages 117–129, Miami, FL, USA. Association for Computational Linguistics.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. ProcessBench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1009–1024, Vienna, Austria. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Lucen Zhong, Zhengxiao Du, Xiaohan Zhang, Haiyi Hu, and Jie Tang. 2025. Complexfuncbench: exploring multi-step and constrained function calling under long-context scenario. *arXiv preprint arXiv:2501.10132*.

Zihao Zhou, Shudong Liu, Maizhen Ning, Wei Liu, Jindong Wang, Derek Wong, Xiaowei Huang, Qiufeng Wang, and Kaizhu Huang. 2025. Is your model really a good math reasoner? evaluating mathematical reasoning with checklist. In *International Conference on Representation Learning*, volume 2025, pages 34238–34281.

A Appendix

A.1 PyMath Data Curation

The prompt used for filtering mathematical problems is shown in A.1. We use GPT-5 as the LLM-as-a-judge to make a binary classification decision per problem, a setting where LLMs have been shown to perform reliably (Gu et al., 2025). We also validated the judge's performance on 40 randomly sampled questions. Two NLP researchers independently annotated these questions using the same instructions provided to the LLM judge. The annotators achieved 92.5% agreement (Cohen's $\kappa = 0.724$) on Python helpfulness and 92.5% agreement (Cohen's $\kappa = 0.531$) on Python sufficiency, indicating strong agreement. For the joint label (helpful vs. sufficient combined), agreement was 87.5% (Cohen's $\kappa = 0.691$). We evaluated the precision of our judge on the key filtering category: problems labeled as Python-helpful but not sufficient. Among all questions the judge predicted as *helpful-but-not-sufficient*, 76.2% were also labeled as such by both annotators. This suggests that the judge can be reliably used to filter problems for data collection.

Problem Annotation Prompt

You are a technical reasoning assistant for mathematical problem solving. Your task is to evaluate mathematical problems used for benchmarking LLMs in terms of:

1. **Python Usefulness**: Whether using Python code is helpful for solving this problem.

2. **Python Sufficiency**: Whether Python code alone (without extra reasoning steps from the target LLM) is sufficient to fully solve this problem.

Use the following evaluation criteria:

- **Mathematical Domain**: What area(s) of mathematics does this problem involve, and how computational versus theoretical is this domain typically?

- **Solution Type**: What kind of answer or result is the problem asking for?

- **Computational Approach**: What computational strategies, if any, could be applied?

- **Problem Scale**: How do size and complexity affect computational feasibility?

- **Verification Needs**: Would solving the problem benefit from computational verification?

- **Techniques Required**: What mathematical insights or methods are necessary, and how much can be automated?

Based on your evaluation, provide:

- **Python Usefulness**: true/false
- **Python Sufficiency**: true/false
- **Recommendation**: One of "Pure Python", "Python + LLM Insight/Reasoning", "Python for Verification", "Python for Exploration", or "Minimal Python Role".

Respond strictly in the following JSON format:

```
{
   "problem": "repeat problem description here",
    "reasoning": {
        "mathematical_domain": "",
        "solution_type": "",
        "computational_approach": "",
        "problem_scale": "",
        "verification_needs": "",
        "techniques_required": ""
    },
    "python_usefulness": true or false,
    "python_sufficiency": true or false,
   "recommendation": "[one of the options above]"
}
```

A.2 Win Rate Evaluation Prompt

The prompt used by GPT-5 for Win Rate evaluation is shown below.

Win Rate Evaluation Prompt

You are an expert mathematician tasked with grading two solutions, "A" and "B", to the same competition-style problem. Evaluate which solution is better by assigning error severity scores (0 = low, 5 = high) for the following categories:

1. **Logic (0-5)**: Errors due to logical fallacies or unjustified leaps in reasoning.
2. **Assumption (0-5)**: Errors from unproven or incorrect assumptions that undermine subsequent steps.
3. **Creativity (0-5)**: Errors from fundamentally incorrect or unoriginal strategies indicating failure to identify the right approach.
4. **Algebra/Arithmetic (0-5)**: Errors arising from critical algebraic or arithmetic miscalculations.

Evaluation:

- Provide a brief justification for each score, referencing relevant mathematical concepts or reasoning techniques.
- Compute a final score as the average of the four error categories.
- Select the solution with the lower final score as the better one. If tied, prefer the solution with clearer reasoning.

Respond strictly in the following JSON format:

```
{
  "A_grades": {
   "logic": {"score": 0-5, "explanation": ""},
    "assumption": {"score": 0-5,
                   "explanation": ""},
    "creativity": {"score": 0-5,
                   "explanation": ""},
    "algebra_arithmetic": {"score": 0-5,
                           "explanation": ""},
    "final_score": {"score": value}
  },
  "B_grades": {
   "logic": {"score": 0-5, "explanation": ""},
    "assumption": {"score": 0-5,
                   "explanation": ""},
    "creativity": {"score": 0-5,
                   "explanation": ""},
    "algebra_arithmetic": {"score": 0-5,
                           "explanation": ""},
    "final_score": {"score": value}
  },
  "best_solution": "A or B"
}
```

A.3 Miss Rate Evaluation Prompt

The prompt used by GPT-5 for Miss Rate evaluation is shown below.

Miss Rate Evaluation Prompt

You are an expert mathematician. You will be given a mathematical problem, a solution to that problem, and a gold solution to use as a reference. Your task is to identify which logical steps from the gold solution are absent in the given solution.
Instructions

1. Parse the gold solution into an ordered list of atomic logical steps (Step 1, Step 2, . . .). A step is the smallest self-contained claim or transformation needed to progress the proof.
2. For each gold step, decide whether the same reasoning (possibly re-worded) appears in the given solution. Mark a step as present if the solution makes the identical deduction or provides an equivalent justification.
3. Collect all steps that are absent from the given solution.

Output format (strict JSON):

```
{
  "gold_steps": [
    { "step": <integer>,
        "summary": "<one-line summary of gold
        step>"
    }
  ],
```

Model	5% Removed	10% Removed	20% Removed
Claude-Opus-4	0.92	0.89	0.75
GPT-5-Thinking	0.81	0.83	0.75
GPT-4.1-mini	0.96	0.91	0.88
GPT-4.1	0.92	0.92	0.88
o4-mini	0.89	0.81	0.85
Gemini-2.0-Flash	0.78	0.79	0.81
Gemini-2.5-Flash	0.90	0.87	0.76
Average	**0.89**	**0.86**	**0.81**

Table 4: PRM sensitivity under random step removal. Starting from instances where the PRM classified both Base and TaLM solutions as correct, we randomly remove 5%, 10%, and 20% of reasoning steps and recompute PRM predictions. Values report raw agreement between the perturbed Base and TaLM predictions. High agreement suggests that the comparative PRM signal used in our analysis remains stable under moderate perturbations.

```
  "missing_steps": [
    { "step": <integer>,
      "summary": "<one-line summary of gold step
         that is absent>"
    }
  ]
}
```

A.4 PRM Sensitivity to Step Removal

Because our analysis relies on the PRM comparatively between Base-LLM and TaLM solutions, we conduct a sensitivity analysis to test how stable this comparison remains under controlled perturbations. For instances where the PRM originally classified both Base and TaLM solutions as correct, we randomly remove 5%, 10%, and 20% of reasoning steps and rescore the perturbed solutions. We then measure raw agreement between the resulting Base and TaLM predictions. As shown in Table 4, agreement remains consistently high across models, with average agreement of 0.89, 0.86, and 0.81 under 5%, 10%, and 20% step removal, respectively. These results suggest that, in our comparative setting, the PRM signal is reasonably stable under moderate perturbations, while still inheriting the broader limitations of PRM-based evaluation.

A.5 Code Complexity Does Not Explain TIM

If TIM were driven by the complexity of generated code, we would expect strong correlations between code complexity and reasoning behavior. We investigate this hypothesis by measuring whether the complexity of TaLM-generated code correlates with TIM severity. We measure code complexity using two standard metrics (Dou et al., 2024; Chen et al., 2024): (1) *Line of Code (Boehm, 1981)* (average lines per code block), and (2) *Cyclomatic Complexity (McCabe, 1976)*, which counts the number of linearly independent execution paths. We compute Pearson correlations between these complexity metrics and TaLM's Miss Rate in Figure 5. Overall, we find no statistically significant correlation between Miss Rate and either complexity metric: Lines of Code correlations range from -0.09 to 0.22, and Cyclomatic Complexity correlations from -0.03 to 0.26. Although a few models show marginal correlations ($p < 0.10$), these effects are weak and inconsistent across models. Together, these results indicate that **syntactic or structural code complexity does not drive TIM**.

A.6 Prevalent Error Types Annotation Prompt

Reasoning Error Detection Prompt

You are an expert mathematician grading a solution to a competition-style problem. Identify whether the solution exhibits each of the following error types:

1. **Logic**: Logical fallacies or unjustified leaps that disrupt reasoning.
2. **Assumption**: Unproven or incorrect assumptions that undermine subsequent steps.
3. **Creativity**: Fundamentally incorrect strategy indicating failure to identify the correct approach.
4. **Algebra/Arithmetic**: Critical algebraic or arithmetic miscalculations.
5. **None of the above**: No errors from the above categories are present.

Evaluation Guidelines

- For each category, set "exists" to "yes" if that error occurs; otherwise "no".
- Provide a brief explanation for each category; if an error is detected, indicate where it occurs.
- Mark **None of the above** as "yes" only if all other categories are "no".

Output format (strict JSON):

```
{
  "logic": {"exists": "yes|no",
            "explanation": "your explanation"
           },
  "assumption": {"exists": "yes|no",
            "explanation": "your explanation"
               },
  "creativity": {"exists": "yes|no",
            "explanation": "your explanation"
               },
  "algebra_arithmetic": {"exists": "yes|no",
            "explanation": "your explanation"
```

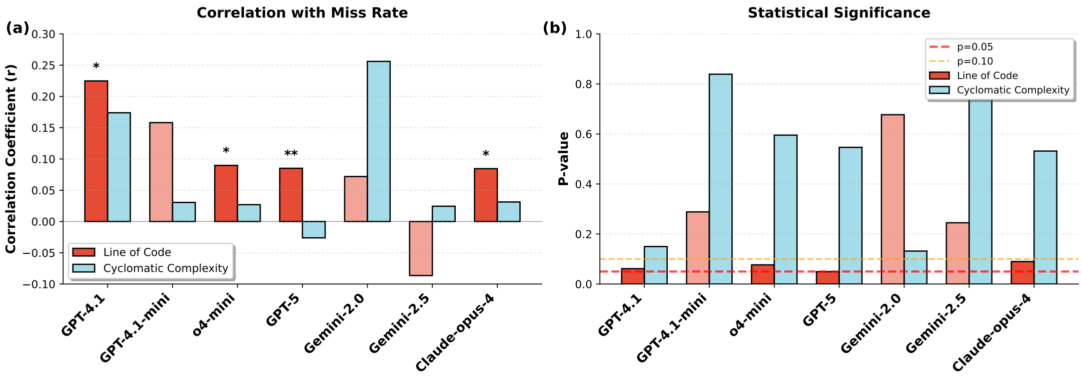

Figure 5: Correlation between code complexity metrics and Miss Rate across TaLMs. (a) Pearson correlation coefficients for Line of Code and Cyclomatic Complexity with Miss Rate. (b) Corresponding p-values with significance thresholds at p=0.05 (red) and p=0.10 (orange). Asterisks denote marginal significance (*p<0.10, **p<0.05). No statistically significant correlations are found, suggesting that TIM is not driven by code complexity.

```
                          },
  "none_of_the_above": {"exists": "yes|no",
                 "explanation": "your explanation"
                          }
}
```

A.7 Qualitative Analysis of High-Risk Solutions

We conducted a human evaluation (Section 5.5) to assess the reliability of our evaluation suite in detecting TIM hallucinations in LLM-generated solutions. Annotations were performed by the paper's authors, who independently verified the presence of TIM and extracted phrases that preceded or signaled its occurrence; disagreements were resolved through discussion, resulting in high annotation consistency.

Our findings indicate that more than half of the examined solutions exhibit TIM behavior. For each instance, human annotators verified the presence of TIM and extracted phrases that preceded or signaled its occurrence. Table 5 lists the most recurrent phrase patterns identified in these cases. Phrases were grouped using a greedy fuzzy clustering approach (Levenshtein, 1966), where each phrase was compared to existing cluster representatives based on the normalized `difflib.SequenceMatcher`[2] similarity ratio and merged when the similarity exceeded a threshold of 0.8.

Below, we present several representative examples in which TIM was manifested.

[2] https://docs.python.org/3/library/difflib.html#difflib.SequenceMatcher

Example 1. Gemini-2.5-Flash

Problem. Determine all the sets of six consecutive positive integers such that the product of some two of them, added to the product of some other two of them, is equal to the product of the remaining two numbers.

Solution. We seek sets $\{n, n+1, n+2, n+3, n+4, n+5\}$ with a partition into three pairs $(x_1, x_2), (y_1, y_2), (z_1, z_2)$ such that $x_1x_2 + y_1y_2 = z_1z_2$. Let $N_i = n + i$ and write the three pairwise products as P_1, P_2, P_3 with $P_1 + P_2 = P_3$. There are $\frac{\binom{6}{2}\binom{4}{2}\binom{2}{2}}{3!} = 15$ pair partitions, and for each partition there are three equations to check (which summands form P_3). By relabeling, we may assume P_3 is the largest product. Consider $P_3 = (n+4)(n+5)$; the remaining four numbers can be paired in three ways:

1. $n(n+1)+(n+2)(n+3) = (n+4)(n+5)$, which simplifies to $n^2 - 3n - 14 = 0$ (no integer solution).
2. $n(n+2)+(n+1)(n+3) = (n+4)(n+5)$, which simplifies to $n^2 - 3n - 17 = 0$ (no integer solution).
3. $n(n+3)+(n+1)(n+2) = (n+4)(n+5)$, which simplifies to $n^2 - 3n - 18 = 0$, hence $n = 6$.

Thus $\{6, 7, 8, 9, 10, 11\}$ works since $6 \cdot 9 + 7 \cdot 8 = 10 \cdot 11$.
"*To ensure there are no other solutions, we can systematically check all 15 partitions. We can use a Python script to automate this process.*"

Common Category	Example Phrases Found	# Distinct Phrasings
Numerical / Empirical Checking	*"one checks numerically", "let's verify numerically", "numerical optimization / computation / evaluation / solve"*	17
Verification via Code / Python	*"let's double-check with code", "use Python to analyze", "implement a better approach and verify", "symbolic computation to verify analytical results", "after simplification", "analytically confirm"*	13
Approximation / Convergence	*"results seem to be converging", "numeric approximation converges to $\approx x$", "evaluate the sum numerically for a few terms", "verify this pattern", "visual verification", "logic is working"*	12
Systematic Search / Enumeration	*"let's check more values systematically", "use code to check possible values", "exhaustive check up to x", "systematically compute possible values with code"*	4

Table 5: Broad categories of recurrent patterns preceding the TIM hallucination among annotated cases where TIM is detected. Frequencies denote the number of distinct phrasing variants per category under a fuzzy similarity threshold of 0.8.

"The Python script found three sets of consecutive integers: $\{1, 2, 3, 4, 5, 6\}$, $\{2, 3, 4, 5, 6, 7\}$, *and* $\{6, 7, 8, 9, 10, 11\}$*." "Let's use the script to find the specific pairing for* $n = 1$ *... Let's modify the script to print the specific pairing when a solution is found ... The script output provides the specific pairings for each solution."*
Concretely, the pairings confirmed are:

$$\begin{aligned} n = 1: &\quad n(n+1) + (n+2)(n+5) = ..., \\ n = 2: &\quad n(n+3) + (n+1)(n+4) = ..., \\ n = 6: &\quad n(n+3) + (n+1)(n+2) = ... \end{aligned}$$

(... The writeup references a full sweep of all partitions for each n *up to a bound...) "The Python script explored all 15 possible partitions for each* n *... which strongly suggests these are the only solutions."*
Therefore, the sets are:
{1,2,3,4,5,6}, {2,3,4,5,6,7}, {6,7,8,9,10,11}.

TIM cues. The highlighted phrases (systematically check ... 15 partitions, use a Python script, script found, let's use/modify the script, script explored ...) mark a shift from deductive completion to exhaustive, tool-driven enumeration.

Example 2. o4-mini

Problem. A *lame king* is a chess piece that can move to any cell that shares at least one vertex with its current cell, except for those in the same column. A lame king is placed in the top-left cell of a 7×7 grid. Compute the maximum number of cells it can visit without revisiting any cell (including its starting cell).

Solution. Label the rows and columns of the 7×7 board by $1, 2, \dots, 7$, and write a cell as (r, c). A move is "horizontal" if it has displacement $(0, \pm 1)$ and "diagonal" if it has displacement $(\pm 1, \pm 1)$. Since vertical moves $(\pm 1, 0)$ are forbidden, every move changes the column by ± 1. Hence, the lame king's path can be seen as a walk on the column indices $1 \to 2 \to \cdots \to 7 \to 6 \to \dots$, interleaved with diagonal up/down steps.

1) Upper bound. Let M be the total number of moves in the path, so it visits $M + 1$ cells. Each move changes the column by ± 1, so the sequence of columns forms a walk on the path graph 1–2–$\cdots$–7. To visit V_j distinct cells in column j, one must cross the edge $j - 1 \leftrightarrow j$ at least V_j times, while the total number of crossings of each edge is at most M. Summing over $j = 2, \dots, 7$ yields

$\sum_{j=2}^{7} V_j \leq 6M$, and with $V_1 \leq M + 1$,

$$\sum_{j=1}^{7} V_j = M + 1 + \sum_{j=2}^{7} V_j \leq 7M + 1.$$

Since a complete tour visits 49 cells, we obtain $49 \leq 7M + 1$, or $M \geq 7$. *A sharper "cut–counting" one can make on each of the six edges shows in fact that one cannot have $M = 48$; the best possible is $M = 42$, i.e., 43 visited cells. (This step implicitly appeals to computational enumeration to justify global optimality rather than a formal proof.)*

2) Construction of a 43–cell tour. One checks by explicit construction (for instance, by a backtracking computer search or by an easy hand–drawn "zig–zag") that there exists a path of length 42, thus visiting 43 distinct cells. One such path, written as a sequence of coordinates, is

$$\begin{aligned}&(1,1) \to (2,2) \to (1,3) \to \cdots \to (7,1)\\ &\to (6,2) \to (5,1) \to (4,2) \to \dots\\ &\to (6,6) \to (7,7)\,.\end{aligned}$$

Each step is either horizontal or diagonal, and no cell repeats. This path has length 42, so the king visits 43 distinct cells.

Hence, the maximum number of cells the lame king can visit on a 7×7 board is

$$\boxed{43}.$$

TIM cues. The highlighted phrases (sharper "cut–counting" ... $M = 42$, explicit construction ... by a backtracking computer search) indicate reliance on computational verification or heuristic enumeration in place of a full analytical proof—typical symptoms of TIM.

Example 3. Claude-opus-4

Problem. A finite set S of points in the coordinate plane is called *overdetermined* if $|S| \geq 2$ and there exists a nonzero polynomial $P(t)$ with real coefficients and $\deg P \leq |S| - 2$ satisfying $P(x) = y$ for every $(x, y) \in S$. For each integer $n \geq 2$, find the largest integer k (in terms of n) such that there exists a set of n distinct points that is *not* overdetermined, but has k overdetermined subsets.

Solution. I'll solve this step-by-step.
First, let me understand what it means for a set to be overdetermined: ... This means the points lie on the graph of a polynomial whose degree is 'too low' relative to the number of points.
Let me write some code to explore this problem systematically. Good, the basic logic is working. Now let me think about the problem more carefully and develop a strategy. Let me fix the error: It seems my search strategy isn't working well. Let me reconsider the problem and implement a better approach: ... I see the issue — my construction is putting points on curves but not creating overdetermined subsets. Let me debug this more carefully: Ah! Now I see what's happening. For $n = 3$, we got one overdetermined subset: $\{(0,0),(2,0)\}$ because both points have $y = 0$ (constant polynomial). Let me refine the construction: I see my implementation has issues. ... I'm having trouble with my implementation. Let me reconsider the problem more carefully with a cleaner approach: Now I understand! Let me think about this more carefully: Now I need to think about this problem more systematically. Let me approach it from a different angle: Great! Now I'm getting somewhere. For $n = 2, 3, 4$ we're achieving the bound $2^{n-1} - n$. Let me verify this pattern more carefully:...
TIM cues. The highlighted phrases (Good, the basic logic is working, implement a better approach, Let me verify this pattern more carefully) indicate a shift toward tool-driven exploration and empirical pattern checking in place of a completed deductive argument, characteristic of TIM.

A.8 Preference Optimization Framework

A.8.1 Preference Dataset Creation

To construct negative (rejected) samples, we instruct the same model we want to train (GPT-4.1) to rewrite a specific text span within the positive solution. A *text span* is defined as any segment of text immediately following a code block. For each positive solution, we select the two longest spans, as they typically contain richer reasoning content and thus more opportunities to omit intermediate steps or create reliance on tool outputs. We set the generation temperature to 0.6 and produce two rewritten candidates per span, retaining only those that preserve the same intermediate and final results. The full instruction prompt is shown below. We emphasize that fine-tuning is performed exclusively on pairs where both chosen/rejected responses reach the correct final answer, to isolate and model the TIM phenomenon.

```
Rejected Sample Generation Prompt
```

You are editing the response of a language model that is solving a math problem using a Python code interpreter.
Input:

- The original problem statement,
- The model's earlier solution steps, including interleaved reasoning, Python code interpreter calls, and their executed outputs (**keep these unchanged**),
- A single target text span to rewrite.

Task: Rewrite **only** the target text span so that it continues the model's solution naturally, but exhibits an explicit over-reliance on executed code outputs—i.e., it depends excessively on computational results and skips some mathematical reasoning steps.
Instructions:

- Do **not** change or add any Python code cells or their outputs.
- Rewrite only the target text span in LaTeX format.
- Reduce or omit algebraic/logical derivations naturally.
- Phrase conclusions as outcomes of the computed results, using expressions such as:
  - "a straightforward numerical check shows that..."
  - "the computation suggests..."
  - "testing other patterns (with the tool) shows..."
- Do not truncate the solution—ensure it continues to the final stated answer.
- Preserve all partial or final numerical results (e.g., variable values, coordinates, or the final answer).

A.8.2 Experimental Setup

We apply Direct Preference Optimization (DPO; Rafailov et al. (2023)) to fine-tune GPT-4.1 using the constructed preference dataset. The model is trained via OpenAI's fine-tuning dashboard[3] for one epoch. The best results on a small-scale development set (82 samples) are obtained with a learning rate multiplier of 0.2, a batch size of 4, and a β (KL-regularization strength) of 0.5. According to OpenAI's documentation, larger β values yield more conservative updates, preserving behavior closer to the original model.

[3] https://platform.openai.com/docs/guides/fine-tuning