Evaluating SageMath-Augmented LLM Agents for Computational and Experimental Mathematics

Pavel Snopov [1 *] **German Magai** [2 *]

Abstract

Recent advances in AI for Mathematics have focused largely on autoformalization and theorem proving, leaving the role of Computer Algebra Systems (CAS) in agentic LLM workflows underexplored. We propose a ReAct-style agentic setup that combines LLM reasoning with verifiable feedback from SageMath, together with Context7 for the up-to-date documentation. We evaluate this agentic setup across frontier models for solving research-level mathematical problems from the RealMath benchmark in a setting that emulates a computational-mathematics research loop. We also propose a refinement to the RealMath benchmark by introducing a multi-step post-processing procedure and a multi-stage validation pipeline, both of which improve the quality and reliability of the extracted problem set. Our experiments reveal substantial performance gains from SageMath access across all evaluated models with OpenAI's flagship model, GPT-5.5, achieving the highest solve rate of 75.2% and the lowest token usage among tool-enabled configurations. Our findings suggest that CAS-augmented agents represent a promising direction for assisting mathematicians in computational exploration, and we believe that this work is a step towards automated conjecture discovery. The project repository is available online. [1]

[*]Equal contribution [1]School of Mathematical and Statistical Sciences, The University of Texas Rio Grande Valley, USA [2]Noeon Research, Tokyo, Japan. Correspondence to: Pavel Snopov <paul.snopov@utrgv.edu>, German Magai <german@noeon.ai>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

[1]https://github.com/Snopoff/Evaluating-SageMath-Augmented-LLM-Agents-for-Computational-and-Experimental-Mathematics

1. Introduction

Recent progress in LLMs and agentic systems that integrate LLM reasoning with deterministic, verifiable tool backends such as compilers, theorem provers, SAT/SMT solvers, type checkers, and physical simulators has established a new neuro-symbolic paradigm in which generative reasoning is partially grounded in verifiable feedback. By coupling LLM reasoning with components that provide ground-truth signals, this paradigm enables a class of tools capable of automating tasks that previously required substantial expert effort and of producing answers with a level of reliability that purely generative approaches cannot achieve.

A particularly active application of this paradigm is in mathematics. Substantial progress has been made on autoformalization, the translation of mathematical statements into the formal programs written in proof-assistant languages such as Lean, Coq, and Isabelle (Wu et al., 2022), as well as on automated theorem proving, where LLM-based systems guide the search for formal proofs (Lin et al., 2025), attain gold-medal-level performance on the 2025 International Mathematical Olympiad with formally verified solutions (Achim et al., 2025), and and have recently autonomously resolved several open problems from the Erdős collection (Sothanaphan, 2026; Tsoukalas et al., 2026) including the well-known unit-distance conjecture (OpenAI, 2026). These developments establish the integration of LLMs with formal proof assistants as a promising direction for mathematical reasoning, focused on proving and formalizing already specified statements. However, this line of work focuses primarily on formal proof construction, leaving the computational mathematics workflow comparatively underexplored.

In many areas of mathematics, Computer Algebra Systems (CAS) and symbolic engines are routinely used for hypothesis exploration, candidate validation, and counterexample search. A common workflow in modern mathematical research, particularly in computational areas such as combinatorial commutative algebra, algebraic combinatorics, graph theory, finite group theory, and related fields, relies heavily on CAS. Prominent examples include SageMath (The Sage Developers, 2024), Macaulay2 (Grayson & Stillman), GAP (The GAP Group, 2024), Magma (Bosma et al., 1997),

and Singular (Decker et al., 2024) which provide computational support for research-level mathematical problem solving. Researchers use such systems to discover and prune the conjecture space by iteratively computing examples, identifying patterns, formulating tentative conjectures, and testing them computationally[2]. Such workflows are particularly common in computational and experimentally driven areas of mathematics.

Despite the widespread use of CAS, their integration into LLM-based systems for mathematical reasoning has received substantially less attention than the integration of formal proof assistants. Many existing approaches focus primarily on formal proof construction, while comparatively little attention has been devoted to computational workflows in mathematics centered around symbolic experimentation and executable verification. This gap, together with the recent successes of LLMs in mathematical reasoning, motivates the present study. We evaluate frontier LLMs within an agentic setup equipped with SageMath-based CAS access that closely emulates a mathematician's computational workflow. The proposed setup is particularly relevant for tasks where symbolic computation plays a central role and where core mathematical structures have already been algorithmically formalized within existing software systems.

Our contributions are threefold: **Tool-use agentic evaluation:** we present an empirical study of frontier LLMs in a ReAct-style agentic setup with multi-turn access to SageMath together with up-to-date documentation retrieval through Context7. We evaluate how verifiable feedback from a CAS affects model performance on research-level mathematical problems through a controlled tool-access ablation. **Benchmark refinement and evaluation protocol:** building on the RealMath benchmark, we curate an evaluation set restricted to numerical and symbolic-expression answers. We additionally enrich problem statements with compressed contextual information and introduce an LLM-assisted normalization pipeline for executable symbolic verification. We further employ a hybrid validation protocol combining symbolic equivalence checking with LLM-as-a-Judge re-evaluation. **Error and behavioral analysis:** we complement the quantitative evaluation with an analysis of execution failures, recovery dynamics, and tool-call traces, highlighting model-specific patterns in how agents interact with symbolic computation tools.

2. Related Work

2.1. Autoformalization and Theorem Proving

This subsection situates our work within the broader recent progress in AI4Math, with a focus on autoformalization and automated theorem proving. The dominant line couples LLMs with formal proof assistants. Many works evaluate LLMs on competition mathematics and formal proof generation. MATHARENA (Balunović et al., 2025) evaluates LLMs on math competitions immediately after they occur to avoid contamination, and introduces a proof-writing benchmark on which top models score below 25% on USAMO 2025. PUTNAMBENCH (Tsoukalas et al., 2024) proposes 1,692 task formalizations across Lean 4, Isabelle, Coq.

HERALD (Gao et al., 2025) builds a large NL-to-Lean 4 corpus by back-translating Mathlib and trains an autoformalizer reaching 96.7% accuracy on miniF2F-test. ALPHAPROOF (Google DeepMind AlphaProof and AlphaGeometry teams, 2025) combines Lean with AlphaZero-style RL, and together with the geometry-specialized ALPHAGEOMETRY 2 (Chervonyi et al., 2025) reached silver-medal performance at IMO 2024. The DEEPSEEK-PROVER line (Xin et al., 2024; Ren et al., 2025) pursued open-source provers through subgoal decomposition, with V2 reaching 88.9% on MiniF2F-test. KIMINA-PROVER (Wang et al., 2025a) and SEED-PROVER (Chen et al., 2025) pushed alternative recipes based on structured reasoning and lemma-style proofs with iterative Lean feedback. More recent work has moved beyond standard proof-search training. DEEPSEEKMATH-V2 (Shao et al., 2025) trains a generator–verifier pair that rewards reasoning steps, scoring 118/120 on Putnam 2024. Agentic systems orchestrating LLMs around the Lean compiler have also emerged: AXIOMPROVER (Axiom Math Team, 2025) produced fully verified Lean proofs for all Putnam 2025 tasks. We refer to Wang et al. (2025b) for a broad survey of LLMs for mathematical reasoning, especially with tool usage.

2.2. Benchmarking AI4Math

Research-level benchmarks. A major line of benchmarks targets mathematics drawn from research-level tasks and literature rather than competitions. FRONTIERMATH (Glazer et al., 2024) pushes the difficulty ceiling with hundreds of unpublished problems crafted by leading mathematicians, including Terence Tao, spanning many research-adjacent areas. By mid-2026, the best reasoning LLMs reach over 50% on Tiers 1–3 and 25-40% on Tier 4 (Epoch AI, 2026). ETERNALMATH (Ma et al., 2026) pursues a fully automated theorem-grounded pipeline that turns recent papers into executable, verifiable tasks, with the goal of producing a continuously refreshable benchmark. An especially relevant work is REALMATH (Zhang et al., 2025), which proposes an automated data-collection pipeline that extracts research-level problems from arXiv and Math Stack Exchange and presents them as (context, question, answer) triples, with a focus on constructive theorems. Our work builds on it, we use the computable subset of the dataset extracted by their pipeline, and also propose refinements to the benchmark.

[2] As famously remarked by V. I. Arnold, "mathematics is the part of physics where experiments are cheap."

Benchmarks involving symbolic computation. The benchmarks closest to our work involve symbolic or CAS-augmented evaluation. IMPROOFBENCH (Schmitt et al., 2025) evaluates LLMs in multi-turn setup with tools (e.g. Python, SageMath, Bash, Web search) on 39 research-level proof-finding problems. It was found out that Grok-4 achieving 23% and GPT-5 26% accuracy. In contrast to our setup, IMPROOFBENCH does not isolate the effect of tool access through an explicit ablation protocol and does not provide a public reference implementation. ASYMOB (Shalyt et al., 2025) comprises 17092 symbolic tasks spanning integration, simplification, limits, and differential equations, with the measuring performance degradation under symbolic perturbations and exposing drops of up to 70%, they used SymPy for answer validation. INTEGRALBENCH (Tang et al., 2025) narrows the scope to 317 graduate-level definite integral problems with symbolic and numerical ground truth, illustrating how tight, domain-restricted benchmarks can isolate specific failure modes.

2.3. Tool-Augmented Agents for Mathematical Discovery and Reasoning

A complementary line of works combines agents with external symbolic and computational tools for the conjecture-space discovery and pruning in mathematical reasoning. FUNSEARCH (Romera-Paredes et al., 2024) and its successor ALPHAEVOLVE (Novikov et al., 2025) traverse the hypothesis space by evolving Python programs scored by a deterministic Python-based evaluator: FUNSEARCH produced new lower bounds on the cap-set problem, and ALPHAEVOLVE generalizes the loop to whole-codebase evolution and reports new SOTA constructions, including an improved scheme for 4×4 complex matrix multiplication. Early work in this direction includes (Chuharski et al., 2024), who sample conjectures about the solubilizer in group theory from LLMs and prune the conjecture space via counterexamples generated by GAP. MATHSENSEI (Das et al., 2024) combines a knowledge retriever, a Python program generator and executor, and the Wolfram-Alpha in a planner-driven pipeline, and ablates these tools on the MATH benchmark to isolate their complementary effects. We generalize this pattern to a general-purpose CAS-based tool and evaluate it on mathematical problem solving.

3. Experimental setup

We evaluate frontier LLMs in controlled tool-free and tool-augmented settings on executable mathematical reasoning tasks. In order to do this, we curate a dataset of research-level mathematical problems, organize both tool-free and tool-augmented agentic pipelines and evaluate their performance via the multi-stage evaluation protocol centered around symbolic verification. The system prompts for both pipelines are provided in Appendices D.1 and D.2. This section describes the curated evaluation suite, the agentic setup, evaluation pipelines, and the validation protocol used throughout the experiments.

3.1. Evaluation Dataset

We extracted a curated subset of 133 problems by performing additional processing steps, multi-stage filtering, and sampling from RealMath benchmark (Zhang et al., 2025), a dataset of research-level mathematical problems that is directly derived from the contemporary literature. The additional steps are crucial, since we focus on executable mathematical reasoning and thus require problems to have algorithmically verifiable solutions. However, RealMath was developed as a Q&A benchmark, in which the `answer` field is represented as a plain LaTeX text, which prevents the direct automatic comparison. Moreover, it was collected without assessing the computational perspective of the problems, thus it might contain problems that are not directly suitable for symbolic computation within the SageMath framework.

We therefore construct the final evaluation suite through the following multi-stage pipeline (see Appendix A.2 for the full details regarding the dataset construction, filtering criteria, and annotation prompts). First, we retain only those problems that can be normalized into SymPy (Meurer et al., 2017) objects that are suitable for automatic verification. Second, we employ an LLM-assisted filtering to focus only on the problems for which the symbolic experimentation or computational verification are feasible within SageMath. Third, we sample from the filtered pool and augment it with additional tasks from underrepresented arXiv categories. We also augment problem statements with the definitions and auxiliary statements that are crucial to interpret the question.

3.2. Agentic Setup

The agentic pipeline is organized to emulate the workflow of a working mathematician. In the tool-free setting, the model solely relies on its internal knowledge resources and reasoning capabilities, that are prone to hallucinations given the nature of the models. However, the reasoning aimed to tackle a complicated mathematical problem might require intermediate computational steps with verifiable feedback. It would allow the agent to refine its reasoning through iterative experimentation and explore the hypothesis space more efficiently.

This pipeline is implemented by wrapping the models in a ReAct-style multi-turn loop (Yao et al., 2023). The agent iteratively cycles through three stages: reasoning about the current state, selecting and invoking a tool (acting), and incorporating the returned observation into the next turn. The available tools are SageMath and Context7.

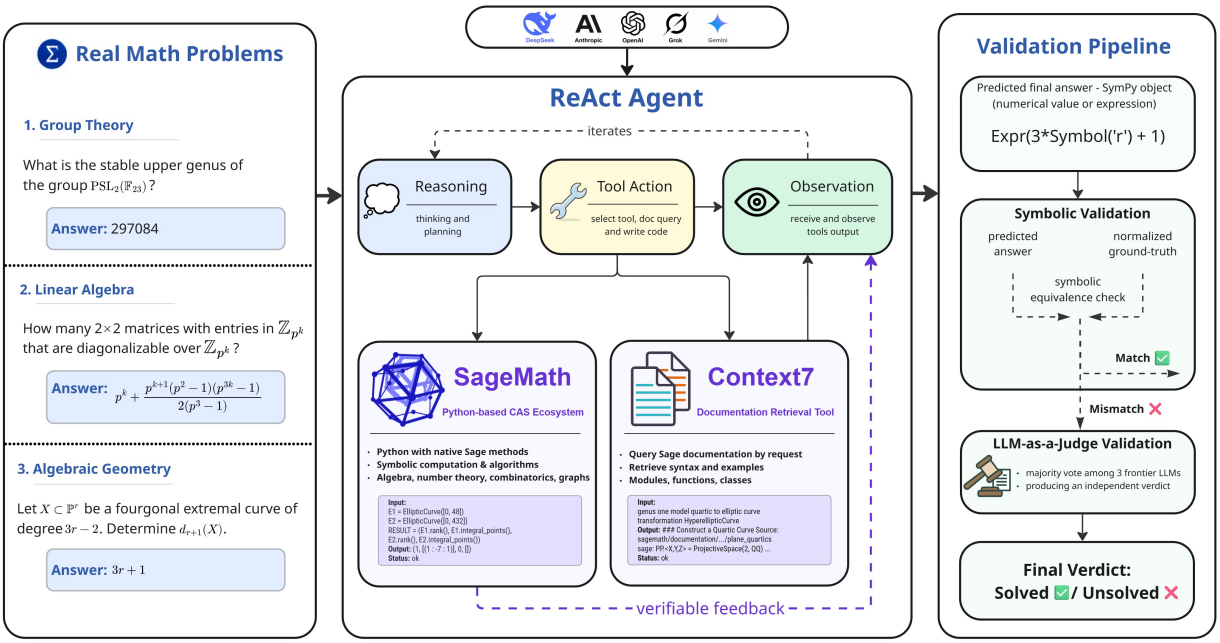

Figure 1. Overview of the proposed evaluation framework for executable mathematical reasoning. A ReAct-style LLM agent interacts with SageMath and Context7 through iterative tool use, receives verifiable feedback from symbolic computation, and produces a final answer validated through symbolic equivalence checking and an auxiliary LLM-as-a-Judge stage.

SageMath. SageMath is a popular open-source computer algebra system that provides a unified Python-based interface to a broad collection of mathematical software systems, such as GAP, Singular, or PARI/GP. It supports symbolic computations, algebraic manipulations, numerical experimentation, and includes many classical algorithms for polynomial ideal computations, graph-theoretic computations, computational number theory and so on. The agent can use this tool for the executable mathematical reasoning by generating Sage code snippets, run them in an isolated sandbox (see Appendix A.1 for details), receive the feedback and iterate on its hypotheses.

Context7. Context7 (Upstash Team, 2026) is a project that provides access to up-to-date documentation and code examples from official sources. The agent is given access to this tool to reduce hallucinations about modules, paths, and outdated API signatures, as the models may rely on outdated information about SageMath APIs acquired during pretraining. Therefore, providing an access to that tool, we mitigate the possibility that models might not know how to write Sage code, focusing on the model's capability for executable, verifiable thinking and its ability to refine candidate solutions.

3.3. Experimental protocol

We evaluate this agentic setup by comparing it with a tool-free baseline to measure the impact of tool integration on performance in research-level mathematics. Both configurations use the same base model, differing only in the ability to interact with external tools. In the agentic setting, we impose a hard cap of 15 tool calls per problem. Additionally, we require the model to return SymPy-parsable output, since the evaluation is based on symbolic verification.

Tool-free setting. The tool-free pipeline consists of direct reasoning performed by a model without external feedback. It allows us to ablate the model's internal capabilities for solving research-level mathematical problems. This setting is similar to standard direct-answer evaluation used in the original RealMath benchmark.

Models and evaluation scope. For the base models, we evaluate both commercial and open-weight frontier models from multiple providers: Claude Opus 4.7 (Anthropic, 2026), DeepSeek-V3.2 (DeepSeek-AI, 2025), DeepSeek-V4-flash (DeepSeek-AI, 2026), OpenAI GPT-5.5 (OpenAI, 2026), Gemini-3.5-Flash (Google DeepMind, 2026b), Gemini-3.1-Pro-preview (Google DeepMind, 2026a), Grok 4.3 (xAI, 2026). accessed via APIs and the unified interface provided by LangChain framework (Chase, 2022). Each model was evaluated under both setups across the dataset of 133 problems. This experimental design yielded a total of more than 1800 independent runs (7 models × 2 setups × 133 problems).

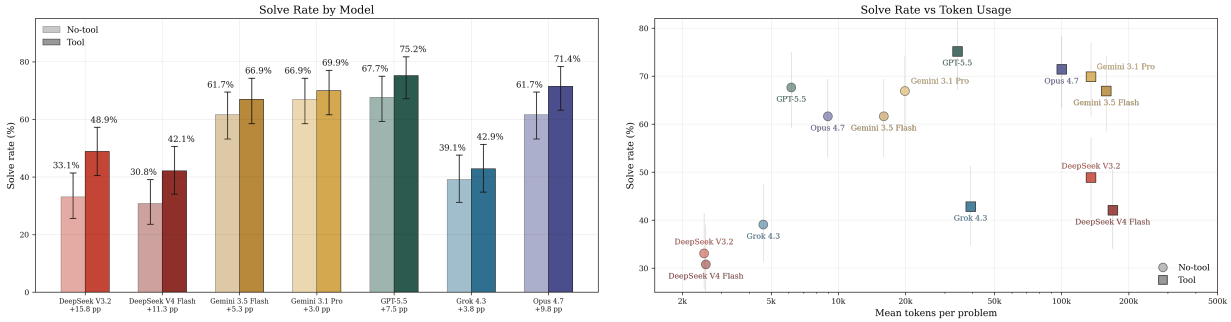

Figure 2. Overall solve rate and token cost across models and evaluation setups. Left: solve rates for each model without tools and with the tool-enabled workflow, error bars show 95% Wilson confidence intervals. Right: solve rates plotted against mean token usage per problem on a logarithmic axis, showing the accuracy-cost trade-off introduced by tool use agentic setup.

3.4. Evaluation pipeline

Validating the equivalence of symbolic mathematical expressions is substantially more challenging than validating numerical answers, since structurally different expressions may still be mathematically equivalent. Prior benchmarks address this issue either through symbolic equivalence procedures (Shalyt et al., 2025) or through LLM-based judging (Tsoukalas et al., 2024; Zhang et al., 2025). We combine both approaches in a unified multi-stage evaluation pipeline. This is necessary because symbolic simplification is not exhaustive and may incorrectly reject mathematically equivalent expressions.

Symbolic validator. The first stage compares the SymPy-parsable output produced by the model against the ground-truth answer represented in the same format. Expressions are parsed into SymPy objects and their difference is simplified symbolically. If the resulting expression simplifies to zero, the prediction is considered correct. For multi-component answers, outputs are compared component-wise.

LLM-as-a-Judge validator. Cases rejected by the symbolic validator are not necessarily incorrect. For such cases, we employ a second-stage evaluation procedure based on three frontier LLM judges: Claude Opus 4.7, GPT-5.5, and Gemini 3.5 Flash[3]. Each model independently produces a verdict together with a short justification, and the final decision is obtained through majority voting. In addition, we manually inspect 30% of the cases reaching this stage and treat the resulting human annotations as ground truth. LLM-based judging has known limitations, e.g. self-preference bias, prompt sensitivity, imperfect agreement with human. We therefore restrict its role to cases where symbolic equivalence checking is unable to produce a final verdict.

[3]The judge prompt is shared across models and provided in Appendix D.5.

Evaluation metric. We report solve rate, defined as the fraction of problems for which the model's final answer is accepted by the validation pipeline. A prediction is accepted either directly by symbolic equivalence checking or, when symbolic validation is inconclusive, through majority voting among the three LLM judges. All solve rate estimates are accompanied by 95% Wilson score confidence intervals.

4. Results and discussion

4.1. Main results

We evaluated seven frontier LLMs in both tool-free and tool-augmented settings. The main results are shown in the left panel of Figure 2. Tool access improved the solve rate of every model, although the magnitude of the gains varied substantially across architectures. The weakest baselines, DeepSeek 3.2 (33.1%) and DeepSeek V4 (30.8%), exhibited the largest improvements, reaching 48.9% (+15.8 pp) and 42.1% (+11.3 pp), respectively. However, strong baselines such as Opus 4.7 and GPT-5.5 also benefited substantially from tool access, improving from 61.7% $\rightarrow$ 71.4% and 67.7% $\rightarrow$ 75.2%, respectively. By contrast, Grok 4.3 improved only marginally despite its relatively weak baseline, gaining just +3.8 pp. This suggests that effective tool use depends not only on underlying mathematical reasoning capability, but also on the model's ability to reliably generate, revise, and execute tool interactions.

Solve rate versus token cost. The right panel of Figure 2 reveals a substantial efficiency gap between tool-augmented settings. Tool access increases mean token consumption by factors ranging from 5.6 to 66.7 across models, but the additional compute does not translate into proportional solve-rate gains. The tool-enabled models occupy a broad region of the accuracy–cost plane, ranging from roughly 34k to 170k tokens per problem with solve rates between 42.1% and 75.2%.

The resulting Pareto structure is highly asymmetric. GPT-5.5 forms the dominant frontier point in the tool-augmented setting, simultaneously achieving the highest solve rate and the lowest token usage among tool-enabled configurations. By contrast, several other models consume substantially more tokens while attaining lower solve rates. For example, Opus 4.7 achieves the second-highest solve rate (71.4%) but requires approximately $3\times$ the tokens used by GPT-5.5, indicating that additional reasoning and tool interaction do not necessarily correspond to more effective problem solving. The weakest efficiency regimes appear for DeepSeek V4 and Grok 4.3, which consume more than 35k and 164k tokens per problem, respectively, while remaining below 45% solve rate. More broadly, the figure suggests that the primary bottleneck is not raw reasoning budget, but the efficiency and stability of the agent-tool interaction loop itself.

An additional qualitative pattern visible in this panel is a provider-level clustering effect. Models from the same family occupy nearby regions of the accuracy–cost plane in both the tool-free and tool-augmented settings. In particular, the two Gemini models cluster closely together, as do the two DeepSeek models. This suggests that agentic efficiency characteristics may partially reflect shared architectural choices, post-training procedures, or tool-use policies inherited within a model family.

Effect of answer type. The dataset contains 73 expression-typed and 60 numerical-typed answers. In the tool-free setting, six out of seven models perform substantially better on numerical problems than on expressions, with the largest gaps observed for the two DeepSeek models. Tool access reduces this disparity primarily by improving expression-type performance: models with weak expression baselines gain up to 14–18 pp on expression tasks, whereas models already strong on expressions gain comparatively little. Grok 4.3 is the only model whose expression performance decreases slightly under tool access, consistent with its SyntaxError-dominated failure mode discussed in Section 4.2. These results suggest that symbolic computation particularly benefits tasks requiring canonicalization and manipulation of symbolic expressions. Detailed breakdowns are provided in Appendix B.4.

Problem coverage and uniquely solved items. Across the 133 problems and 14 evaluated configurations, 110 problems were solved by at least one tool-augmented agent, while 20 problems were solved by no configuration. Nine problems were solved by exactly one configuration, with GPT-5.5 accounting for five of these unique solves. The remaining uniquely solved problems were distributed across DeepSeek V4, Gemini 3.5 Flash, Gemini 3.1 Pro, and Grok 4.3. Universally unsolved problems are concentrated in areas requiring substantial theoretical insight or specialized contemporary mathematical knowledge beyond direct symbolic experimentation. Additional qualitative analysis and examples are provided in Appendix B.5.

Solve rate by mathematical category. Tool access improves performance unevenly across arXiv categories. The largest gains appear in combinatorics and algebraic geometry, whereas number theory shows comparatively little improvement. Group theory remains difficult for all models, with solve rates below 45% across both settings, suggesting either limited use or limited effectiveness of SageMath's GAP backend on these tasks. Algebraic topology exhibits the strongest polarization: GPT-5.5 and Opus 4.7 achieve solve rates of around 60% with tools, while several models remain at 0% in both settings. Detailed category-wise results are reported in Appendix B.1.

LLM-as-a-judge validation. The hybrid validation pipeline behaved consistently across both evaluation settings. SymPy resolved 35.9% of tool-free cases and 40.4% of tool-augmented cases directly through symbolic equivalence checking; the remaining predictions were routed to the LLM-based judging stage. Judge-panel consensus remained high in both settings (97.2% for tool-free and 95.3% for tool-augmented runs), with pairwise agreement between judges ranging from 96.4% to 98.5%. Moreover, consensus positive verdicts occurred on 141 tool-free and 170 tool-augmented cases, suggesting that many symbolic false negatives correspond to mathematically equivalent answers expressed in different canonical forms. These results indicate that the LLM-as-a-Judge stage primarily acts as a recovery mechanism for symbolic-verification failures rather than as an independent source of noisy supervision.

4.2. Error analysis and tool-use traces

We next analyze how agents fail when interacting with SageMath. We define a tool-execution error as an unsuccessful call of a Sage tool: generated code either raises an exception, exceeds the 45-second wall-clock limit, or crashes the SageMath process below the Python layer.

Tool-use depth. Figure 3 shows the distribution of agentic tool call trace lengths across models. The distributions are strongly polarized rather than unimodal: most runs either terminate after only a few tool calls or approach the 15-call budget limit. Stronger models such as GPT-5.5 and Opus 4.7 concentrate most of their mass in the 3–4 turn region, suggesting that they frequently require only a small number of computational checks before converging to a final answer. In contrast, weaker models, particularly the two DeepSeek configurations, accumulate substantial mass near the maximum tool budget, indicating prolonged exploratory or recovery behavior before termination. Context7 calls are concentrated

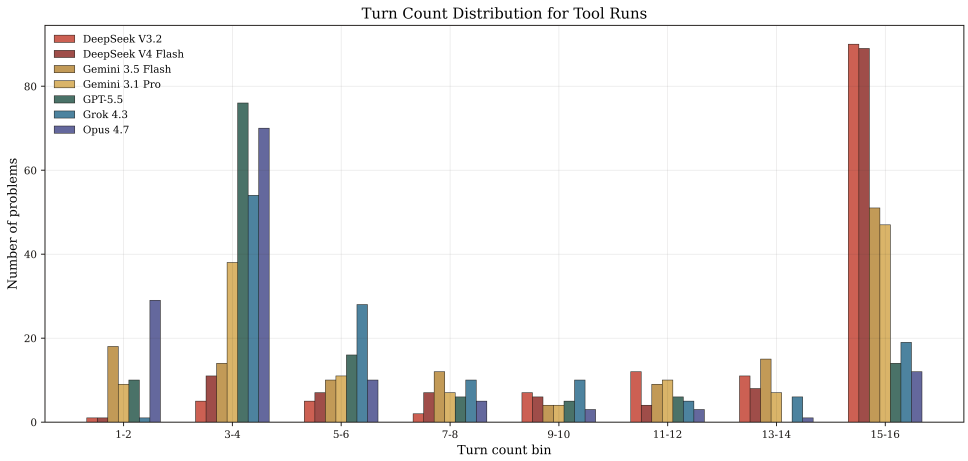

Figure 3. Distribution of agentic trace lengths across models. Each bar shows the number of problems for which the agent terminated after a given number of tool calls.

in the lighter DeepSeek models, while larger models rarely consult documentation, suggesting stronger prior familiarity with the Sage documentation (see Figure 10).

The long-tail behavior differs qualitatively across models. For strong agents, traces exceeding 8–10 tool calls are relatively rare and likely correspond to problems where the model genuinely relies on iterative computational exploration rather than using SageMath merely as a lightweight verifier. By contrast, long traces for weaker models often coincide with repeated execution failures and unsuccessful recovery attempts. This separation also aligns with the accuracy–cost trade-off observed in Figure 2. Models that frequently exhaust large portions of the tool budget are precisely those with the highest token consumption per problem and the weakest efficiency profiles. Conversely, the strongest agents achieve both higher solve rates and shorter traces, indicating that effective agentic performance depends not only on access to external tools, but also on the ability to use them selectively and terminate exploration once sufficient evidence has been obtained.

Sage execution failures. Table 1 summarizes failed Sage tool calls by exception type across models; additional visualizations are provided in Appendix B.2. Failure counts differ sharply across models. DeepSeek 3.2 produces the largest number of execution failures (464), whereas GPT-5.5 and Opus 4.7 produce only 31 and 33. Grok 4.3 exhibits a failure profile dominated by syntactic errors, with `SyntaxError` accounting for the majority of its failed calls.

Across all models, four exception classes dominate: `NameError`, `TypeError`, `SyntaxError`, and `AttributeError`, together accounting for roughly 87% of all failures. `NameErrors` typically arise because each Sage tool call is executed in a fresh interpreter and variables do not persist across turns. `TypeErrors` often reflect mismatches between Sage and native Python object types, while `AttributeErrors` usually correspond to invoking methods on mathematically related but incompatible Sage objects. The remaining exception categories contribute comparatively little.

Table 1. Failed Sage calls by exception type. Top four classes shown, rest grouped as Other.

Model	Name	Type	Syntax	Attr.	Other	Total
DeepSeek 3.2	151	108	11	108	86	464
DeepSeek V4-flash	80	84	14	42	35	255
Gemini 3.5 Flash	99	62	2	50	24	237
Gemini 3.1 Pro	45	45	3	18	28	139
GPT-5.5	4	13	3	6	5	31
Grok 4.3	23	14	248	17	12	314
Opus 4.7	0	21	3	3	6	33
Total	402	347	284	244	196	1473

Recovery after failed calls. The ability to recover after a failed tool call varies substantially across models. The probability that the next Sage tool call succeeds after a failure ranges from 32% for Grok 4.3 to 77% for GPT-5.5, with the remaining models falling between 50% and 71%. Grok 4.3 is the clearest outlier: its low recovery rate is consistent with its SyntaxError-dominated failure profile, since repeated parse failures prevent the agent from converting feedback into a successful computation. DeepSeek 3.2 also recovers poorly, consistent with trace-level behavior in which the model often revisits the same flawed approach across consecutive calls. These results suggest that the main axis separating agents is not simply whether they make tool-use errors, but whether they can revise their strategy after receiving tool feedback.

Trace-level behavior. To inspect agent-tool interaction beyond aggregate error counts, we visualize tool-call traces as tiled sequences, where each tile denotes a successful call, an execution error, a timeout, or another tool response, see Figure 11. The traces reveal a qualitative separation between models. GPT-5.5 and Opus 4.7 typically solve problems with short traces containing one or two successful calls, occasionally recovering from a single failed attempt. By contrast, DeepSeek 3.2, Grok 4.3, and some Gemini traces are longer and contain clusters of repeated errors before either recovering or reaching the tool-call budget. The tiled visualization also shows that failed calls are not independent events: once an agent commits to an incorrect API pattern, malformed Sage construct, or unsuitable computational strategy, it often repeats variants of the same failure before pivoting to an alternative approach. This clustering is most pronounced for DeepSeek 3.2 and least pronounced for Opus 4.7, supporting the interpretation that recovery from tool errors is a central determinant of agentic performance.

4.3. Limitations

The present study has several limitations. First, the symbolic-verification procedure may fail to recognize correct answers, since robust universal equivalence checking remains difficult in the general case. Second, both dataset post-processing and secondary validation rely on LLM outputs, which are not fully reliable. Third, the experimental dataset is limited in size and restricted to numerical or symbolic-expression answers that can be normalized and automatically checked, introducing selection bias toward the computationally executable subset of research-level mathematics. Finally, the tool-augmented setting combines SageMath, Context7, ReAct prompting, and a tool-use budget; thus, gains cannot be attributed to CAS access alone.

5. Case Study: Conjecture Discovery via Tool-Assisted Iterative Computation

In this section, we examine in detail a solution trajectory for a representative problem, illustrating how a CAS-augmented agent can exhibit key elements of a computational mathematician's workflow. Rather than querying SageMath for the target invariants directly, the agent iteratively formulates and tests conjectures, diagnoses errors, and extracts patterns from computational results. Neither invariant is directly implemented as a SageMath routine, requiring the agent to infer the target formulas indirectly based on iterative computation. The problem is drawn from (Himeno & Teragaito, 2024) and focuses on the torsion order $\mathrm{Ord}(K)$ and the second torsion order $\mathrm{Ord}'(K)$ of the twisted torus knot $K = T(p, kp+1; 2, 1)$. Specifically, the task asks for closed-form expressions for $\mathrm{Ord}(K)$ when $p \geq 2$ and for $\mathrm{Ord}'(K)$ when $p \geq 4$. The complete problem statement and background are provided in Appendix C, which also contains concise case studies of other tasks.

Stage 1. Problem setup and capability discovery. Agent started with the exploration of its capabilities within SageMath. It constructed a `BraidGroup` object and inspected whether methods such as `alexander_polynomial` and `burau_matrix` are available.

Stage 2. From braid representatives to staircase complexes. The agent considered the positive braid representation of the knot K

$$(\sigma_1\sigma_2\cdots\sigma_{p-1})^{kp+1}\sigma_1^2,$$

and computed Alexander polynomials for several small values of p with $k = 1$.

Listing 1. Sage computation of small-case Alexander polynomials.

```
def alex_twisted(p, k):
   B = BraidGroup(p)
   gens = B.gens()
   beta = prod(gens)^(k*p + 1) * gens[0]^2
   return beta.alexander_polynomial()

RESULT = [(p, alex_twisted(p, 1)) for p in
    range(2, 7)]
```

Later the agent extracted the exponents of nonzero monomials of the Alexander polynomials and interpreted the gaps between successive exponents as staircase lengths for the associated L-space knot complex. The first implementation attempt failed with a `TypeError` due to inconsistent exponent representations in SageMath, but after adapting the implementation, the revised routine produced:

p	staircase lengths	max
2	$[1,1,1,1]$	1
3	$[1,2,1,1,2,1]$	2
4	$[1,3,1,1,1,1,1,1,3,1]$	3
5	$[1,4,1,1,1,2,1,1,2,1,1,1,4,1]$	4
6	$[1,5,1,1,1,3,1,1,2,2,1,1,3,1,1,1,5,1]$	5
$\vdots$	$\vdots$	$\vdots$

In all computed examples, the largest staircase length was equal to $p-1$. Using the standard staircase description of knot Floer homology for L-space knots, the agent therefore formulated the conjecture

$$\mathrm{Ord}(K) = p - 1.$$

Stage 3. Computing Smith invariant factors. To study the second torsion order, the agent used the staircase lengths obtained from the Alexander polynomial to build an explicit presentation matrix over $\mathbb{F}_2[U]$. For the consecutive

staircase lengths $\ell_0, \ell_1, \ldots, \ell_{2n-1}$, the agent formed the $(n+1) \times n$ matrix M with nonzero entries

$$M_{j,j} = U^{\ell_{2j}}, \qquad M_{j+1,j} = U^{\ell_{2j+1}}, \qquad j = 0, \ldots, n-1.$$

It then computed the Smith normal form of M over $\mathbb{F}_2[U]$. The diagonal Smith entries are the invariant factors of this presentation; when such an entry has degree d, it contributes a U-torsion factor with exponent d.

For $k = 1$ and $p = 2, \ldots, 8$, the largest degrees of the diagonal Smith factors were

$$1, 1, 1, 1, 2, 2, 3.$$

Thus, in the range $p \geq 4$ relevant to $\mathrm{Ord}'(K)$, the computed values agree with

$$\mathrm{Ord}'(K) = \left\lfloor \frac{p-2}{2} \right\rfloor.$$

Stage 4. Testing stability across the parameter k. The preceding computations were carried out with $k = 1$. The agent repeated the extraction of staircase lengths and Smith invariant factors for $k = 2, 3$ and $p = 3, 4, 5, 6$. In these tests, the maximal staircase length remained $p - 1$, and the maximal Smith degree was unchanged as k varied. It supported the hypothesis that both torsion orders depend only on p but not on k.

Attempts to push the computation to larger p eventually ran into SageMath timeouts and low-level runtime failures. At that point, however, the observed formulas had stabilized across the tested values:

$$\mathrm{Ord}(K) = p - 1, \qquad \mathrm{Ord}'(K) = \left\lfloor \frac{p-2}{2} \right\rfloor.$$

Summary. This example illustrates a successful form of tool-augmented mathematical reasoning. SageMath was not used as an oracle for $\mathrm{Ord}(K)$ or $\mathrm{Ord}'(K)$, since neither invariant was directly implemented. Instead, the agent used SageMath to compute accessible intermediate objects: braid Alexander polynomials, exponent gaps, and Smith normal forms over $\mathbb{F}_2[U]$. It recovered from an implementation error caused by Sage's exponent representation, converted Alexander polynomial data into staircase lengths, and used Smith invariant factors to detect U-torsion exponents.

The resulting formulas were not returned by a single Sage command, but were obtained through an iterative process in which the agent computed examples, identified patterns, formulated conjectures, and validated them across other parameters. In this sense, the agent's approach resembles a computational experiment: SageMath supplied evidences, while the agent used its mathematical knowledge to interpret it and infer the conjectural closed forms.

6. Conclusions

We evaluated seven frontier LLMs on 133 research-level mathematical problems derived from the RealMath benchmark, comparing direct-answer inference against a ReAct-style agentic pipeline equipped with SageMath and Context7 documentation retrieval. Tool access consistently improved performance across all evaluated models, with gains ranging from 3.8 to 15.8 percentage points. OpenAI's flagship model, GPT-5.5, achieved the highest solve rate of 75.2% and the lowest token usage among tool-enabled configurations. The largest improvements occurred on symbolic-expression problems requiring executable verification and algebraic manipulation, particularly in combinatorics and algebraic geometry.

Beyond overall solve rates, our experiments reveal that successful agentic mathematical reasoning depends not only on raw reasoning capability, but also on the efficiency and stability of the agent-tool interaction loop. Stronger models achieve higher solve rates while using substantially fewer tool calls and tokens, whereas weaker agents frequently exhaust the tool budget through repeated execution failures and unsuccessful recovery attempts. Our trace-level analysis suggests that recovery dynamics after failed tool calls constitute a central axis separating effective and ineffective tool-augmented mathematical agents.

More broadly, our results support the view that computer algebra systems as a tool provide a promising substrate for computational and experimental mathematics research with LLM agents. While current systems remain limited on problems requiring deep theoretical insight or specialized contemporary knowledge, the observed gains indicate that executable symbolic computation can substantially extend the practical reasoning capabilities of frontier models. We believe this direction represents a promising step toward automated mathematical experimentation and computational conjecture discovery.

Impact Statement

This work studies CAS-assisted LLM agents for research mathematics. It advances verifiable AI-assisted scientific reasoning; we foresee no specific societal risks beyond those already recognized and release our evaluation set and validation pipeline for reproducibility.

Acknowledgments

German Magai gratefully acknowledges Noeon Research for its support during the preparation of this work. Both authors thank Anton Ayzenberg for insightful discussions and comments that helped shape the initial idea and narrative.

References

Achim, T., Best, A., Bietti, A., Der, K., Fédérico, M., Gukov, S., Halpern-Leistner, D., Henningsgard, K., Kudryashov, Y., Meiburg, A., Michelsen, M., Patterson, R., Rodriguez, E., Scharff, L., Shanker, V., Sicca, V., Sowrirajan, H., Swope, A., Tamas, M., Tenev, V., Thomm, J., Williams, H., and Wu, L. Aristotle: IMO-level Automated Theorem Proving. *arXiv preprint arXiv:2510.01346*, 2025. URL https://arxiv.org/abs/2510.01346.

Anthropic. Claude opus 4.7 system card, 2026. URL https://www.anthropic.com/claude-opus-4-7-system-card.

Axiom Math Team. AxiomProver: Solutions to Putnam 2025, 2025. URL https://github.com/AxiomMath/Putnam2025.

Balunović, M. et al. MathArena: Evaluating LLMs on Uncontaminated Math Competitions. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2505.23281.

Barnabei, M., Bonetti, F., Castronuovo, N., and Silimbani, M. Pattern avoiding alternating involutions. *arXiv preprint arXiv:2206.13877*, 2022.

Bosma, W., Cannon, J., and Playoust, C. The Magma algebra system. I. The user language. *Journal of Symbolic Computation*, 24(3–4):235–265, 1997.

Capiński, M. J., Kepley, S., and James, J. M. Computer assisted proofs for transverse collision and near collision orbits in the restricted three body problem. *Journal of Differential Equations*, 366:132–191, 2023.

Chase, H. LangChain, oct 2022. URL https://github.com/langchain-ai/langchain.

Chen, L., Gu, J., Huang, L., Huang, W., Jiang, Z., Jie, A., Jin, X., Jin, X., Li, C., Ma, K., Ren, C., Shen, J., Shi, W., Sun, T., Sun, H., Wang, J., Wang, S., Wang, Z., Wei, C., Wei, S., Wu, Y., Wu, Y., Xia, Y., Xin, H., Yang, F., Ying, H., Yuan, H., Yuan, Z., Zhan, T., Zhang, C., Zhang, Y., Zhang, G., Zhao, T., Zhao, J., Zhou, Y., and Zhu, T. H. Seed-prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*, 2025. URL https://arxiv.org/abs/2507.23726.

Chervonyi, Y., Trinh, T. H., Olšák, M., Yang, X., Nguyen, H., Menegali, M., Jung, J., Verma, V., Le, Q. V., and Luong, T. Gold-medalist Performance in Solving Olympiad Geometry with AlphaGeometry2. *Journal of Machine Learning Research*, 26:1–39, 2025. URL https://arxiv.org/abs/2502.03544.

Christianen, M., Janssen, A., Vlasiou, M., and Zwart, B. Asymptotic analysis of emden–fowler type equation with an application to power flow models. *Indagationes Mathematicae*, 34(5):1146–1180, 2023.

Chuharski, J., Rojas Collins, E., and Meringolo, M. Mining math conjectures from LLMs: A pruning approach. *arXiv preprint arXiv:2412.16177*, 2024.

Das, D., Banerjee, D., Aditya, S., and Kulkarni, A. MathSensei: A tool-augmented large language model for mathematical reasoning. *arXiv preprint arXiv:2402.17231*, 2024. URL https://arxiv.org/abs/2402.17231. Also in Proc. NAACL 2024.

Decker, W., Greuel, G.-M., Pfister, G., and Schönemann, H. SINGULAR *4-4-0 — A computer algebra system for polynomial computations*, 2024. URL https://www.singular.uni-kl.de.

DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

DeepSeek-AI. Deepseek v4 technical report, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Flash/blob/main/DeepSeek_V4.pdf.

Epoch AI. FrontierMath Benchmark Leaderboard. https://epoch.ai/benchmarks/frontiermath-tiers-1-3, 2026. Accessed: 2026-05-24.

Gao, G., Wang, Y., Jiang, J., Gao, Q., Qin, Z., Xu, T., and Dong, B. Herald: A Natural Language Annotated Lean 4 Dataset. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.10878.

Glazer, E. et al. FrontierMath: A Benchmark for Evaluating Advanced Mathematical Reasoning in AI. *arXiv preprint arXiv:2411.04872*, 2024. URL https://arxiv.org/abs/2411.04872.

Google DeepMind. Gemini 3.1 pro model card, 2026a. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/.

Google DeepMind. Gemini 3.5 flash model card, 2026b. URL https://deepmind.google/models/model-cards/gemini-3-5-flash/.

Google DeepMind AlphaProof and AlphaGeometry teams. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 2025. doi: 10.1038/s41586-025-09833-y. URL https://www.nature.com/articles/s41586-025-09833-y.

Grayson, D. R. and Stillman, M. E. *Macaulay2, a software system for research in algebraic geometry*. URL https://www.macaulay2.com.

Himeno, K. and Teragaito, M. Hyperbolic knots with arbitrarily large torsion order in knot floer homology, 2024. URL https://arxiv.org/abs/2412.20652.

Lin, Y., Tang, S., Lyu, B., Yang, Z., Chung, J.-H., Zhao, H., Jiang, L., Geng, Y., Ge, J., Sun, J., Wu, J., Gesi, J., Lu, X., Acuna, D., Yang, K., Lin, H., Choi, Y., Chen, D., Arora, S., and Jin, C. Goedel-Prover-V2: Scaling Formal Theorem Proving with Scaffolded Data Synthesis and Self-Correction. *arXiv preprint arXiv:2508.03613*, 2025. URL https://arxiv.org/abs/2508.03613.

Ma, J. et al. EternalMath: A living benchmark of frontier mathematics that evolves with human discovery. *arXiv preprint arXiv:2601.01400*, 2026.

Martin, G., Salehi, K., and Yamashita, Y. The $(4, p)$-arithmetic hyperbolic lattices, $p \geq 2$, in three dimensions. *arXiv preprint arXiv:2206.14174*, 2022.

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M. J., Terrel, A. R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., and Scopatz, A. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J. R., Mehrabian, A., Kumar, M. P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., and Balog, M. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. URL https://arxiv.org/abs/2506.13131.

OpenAI. Gpt-5.5 system card, 2026. URL https://openai.com/index/gpt-5-5-system-card/.

OpenAI. Planar point sets with many unit distances, 2026. URL https://cdn.openai.com/pdf/74c24085-19b0-4534-9c90-465b8e29ad73/unit-distance-proof.pdf.

Phuc, D. V. On the hit problem for the polynomial algebra and the algebraic transfer. *arXiv preprint arXiv:2412.02494*, 2024.

Ren, Z. Z. et al. DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition. *arXiv preprint arXiv:2504.21801*, 2025. URL https://arxiv.org/abs/2504.21801.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J. S., Wang, P., Fawzi, O., Kohli, P., and Fawzi, A. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Schmitt, J., Bérczi, G., Dekoninck, J., Feusi, J., Gehrunger, T., et al. IMProofBench: Benchmarking AI on research-level mathematical proof generation. *arXiv preprint arXiv:2509.26076*, 2025.

Shalyt, M., Elimelech, R., and Kaminer, I. ASyMOB: Algebraic symbolic mathematical operations benchmark. *arXiv preprint arXiv:2505.23851*, 2025.

Shao, Z., Luo, Y., Lu, C., Ren, Z. Z., et al. DeepSeekMath-V2: Towards Self-Verifiable Mathematical Reasoning. *arXiv preprint arXiv:2511.22570*, 2025. URL https://arxiv.org/abs/2511.22570.

Sothanaphan, N. Resolution of Erdős Problem #728: A Writeup of Aristotle's Lean Proof. *arXiv preprint arXiv:2601.07421*, 2026. URL https://arxiv.org/abs/2601.07421.

Tang, B., Yang, X., Wang, Y., Qiu, Z., Ji, Z., and Jiang, W. IntegralBench: Benchmarking LLMs with Definite Integral Problems. *AI for Math Workshop at the 42nd International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2507.21130.

The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.13.1*, 2024. URL https://www.gap-system.org.

The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 10.5)*, 2024. URL https://www.sagemath.org.

Tsoukalas, G., Lee, J., Jennings, J., Xin, J., Ding, M., Jennings, M., Thakur, A., and Chaudhuri, S. PutnamBench: Evaluating neural theorem-provers on the Putnam mathematical competition. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. arXiv:2407.11214.

Tsoukalas, G., Kovsharov, A., Shirobokov, S., Surina, A., Firsching, M., Bérczi, G., Ruiz, F. J. R., Suggala, A., Wagner, A. Z., Wieser, E., Yu, L., Huang, A., Horváth, M. Z., Ferrauiolo, A., Michalewski, H., Grosu, C., Hubert, T.,

Balog, M., Kohli, P., and Chaudhuri, S. Advancing mathematics research with ai-driven formal proof search, 2026. URL https://arxiv.org/abs/2605.22763.

Upstash Team. Context7: Up-to-date documentation for llms and ai code editors. https://context7.com/, 2026. URL https://context7.com/. Context7 is an Upstash project.

Wang, H., Unsal, M., Lin, X., Baksys, M., Liu, J., et al. Kimina-Prover Preview: Towards Large Formal Reasoning Models with Reinforcement Learning. *arXiv preprint arXiv:2504.11354*, 2025a. URL https://arxiv.org/abs/2504.11354.

Wang, P.-Y., Liu, T.-S., Wang, C., Wang, Y.-D., Yan, S., Jia, C.-X., Liu, X.-H., Chen, X.-W., Xu, J.-C., Li, Z., and Yu, Y. A survey on large language models for mathematical reasoning. *ACM Computing Surveys*, 2025b. doi: 10.1145/3786333. Just Accepted; also available as arXiv:2506.08446.

Wu, Y., Jiang, A. Q., Li, W., Rabe, M. N., Staats, C., Jamnik, M., and Szegedy, C. Autoformalization with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

xAI. Grok 4.3, 2026. URL https://docs.x.ai/developers/models/grok-4.3. Model documentation.

Xin, H. et al. DeepSeek-Prover: Advancing Theorem Proving in LLMs through Large-Scale Synthetic Data. *arXiv preprint arXiv:2405.14333*, 2024. URL https://arxiv.org/abs/2405.14333.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.03629.

Zhang, J., Petrui, C., Nikolić, K., and Tramèr, F. RealMath: A Continuous Benchmark for Evaluating Language Models on Research-Level Mathematics. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2025. URL https://arxiv.org/abs/2505.12575.

A. Experimental Details

A.1. Implementation Details

SageMath Sandbox. All SageMath execution is performed in an ephemeral Docker container. For each tool call, a generated Sage program is launched through a small wrapper. This wrapper imports `sage.all`, runs Sage's preparser to standardize Sage syntax, executes the script and extracts the designated result variable. It also captures standard output and returns a JSON-friendly representation of the result to the host.

For each tool call, the generated code snippet is executed in a fresh interpreter. The execution is limited to 2 CPU cores, 4 GB of memory, a 45s wall-clock timeout, and a 50s CPU-time limit. The executed code has no network access. The host also rejects outputs larger than 262,144 bytes before parsing the returned result to prevent excessively large intermediate outputs.

Failure handling is normalized into a small number of categories. User-code exceptions are returned as `code_error` together with the captured traceback. Docker startup failures, such as an unavailable daemon or missing image, are classified as `docker_error`. Container crashes or abnormal exits are classified as `runtime_crash`, and wall-clock overruns are classified as `timeout`; in the timeout case the host forcibly removes the container.

A.2. Dataset Construction

We begin with the dataset of 633 problems obtained from the RealMath benchmark (Zhang et al., 2025). We focus on this dataset because it is derived from the contemporary mathematical literature. Each problem from this dataset contains the question, context and the answer, each of which is represented in a LaTeX format. We also enrich the dataset enriching it with the arXiv categories that were associated with the papers from which the corresponding problems were scraped via the RealMath extraction pipeline. If the paper had several categories, only the first one was used.

We automatically retain only those tasks whose final answers can be normalized into executable SymPy (Meurer et al., 2017) objects suitable for automatic verification. Specifically, we want the answers to be normalized into either a concrete numerical value, or a symbolic expression that might consist of several free variables or use the widely used mathematical notation (such as π for $3.14159...$, e for $2.71828...$, Γ for the Gamma function, etc). In order to do this, we employ LLM-assisted normalization procedure, under which the original ground truth answer, given in the free LaTeX form, is transformed into SymPy-parsable string. Under this procedure we reduce the problem size in half, resulting in 300 problems. Since the context field in RealMath often contains excessive or irrelevant information, we use an LLM to compress the context, retaining only the relevant definitions and incorporating them directly into the question. We automatically annotate the answer type as one of the following: a number is a concrete numerical value, an expression is a symbolic expression in some free variables, a formula is an equality "named invariant = expression" in which the left-hand side is a label rather than a checkable quantity.

As a next step, we employ an LLM-assisted annotation stage to estimate whether symbolic experimentation or computational verification is feasible within SageMath. In order to do this, we define several criteria for a problem to be suitable for our study: a task is suitable if

1. Objects mentioned in the question can be represented by the means of SageMath,
2. Candidate answers can be explored with the help of computational experiment,
3. Intermediate computation steps can be executed in SageMath,

This stage reduces the problem set to 219. From this, we sample 133 problems attempting to reduce the skewness of the distribution across arXiv categories. Figure 4 illustrates the distribution of the selected problems across corresponding arXiv categories.

B. Additional Empirical Analyses

B.1. Performance by Mathematical Category

Tool access improves performance unevenly across arXiv categories. The largest gains appear in Combinatorics and Algebraic Geometry, whereas Group Theory and Algebraic Topology remain challenging for most models. The strongest

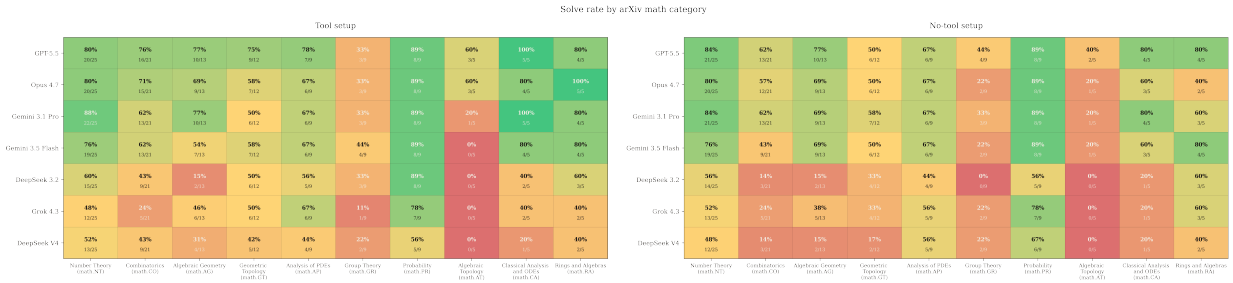

Figure 4. Solve rate by arXiv category with and without tools. Cells show solve rate and solved/total counts.

models, GPT-5.5 and Opus 4.7, exhibit substantially better performance on Algebraic Topology than the remaining configurations.

We report the mean number of tool calls (averaged over all models) against the solve rate for various categories to study how tool usage efforts is related to performance across arXiv categories. Figure 5 visualizes this relationship, with bubble size encoding the number of problems per area.

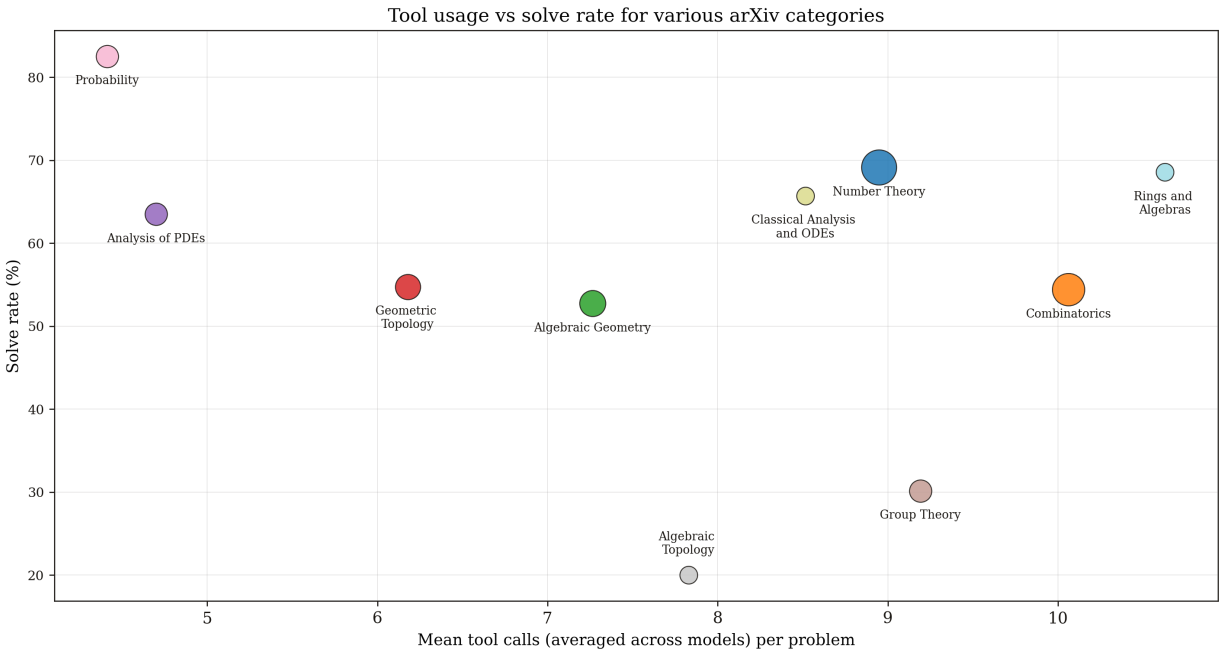

Figure 5. The relationship between the mean number of tool calls and the solve rate for various arXiv categories.

We find no relation between tool usage and accuracy. Probability is solved best with the fewest calls (82.5%, 4.4 calls) and Analysis of PDEs is similar (63.5%, 4.7). Number theory reaches 69.1% at 8.9 calls. But in contrast, Rings and Algebras area (10.6 calls, 68.6%) and Combinatorics (10.1 calls, 54.4%) gain a bit solve rate gain from additional tool calls. The agents struggle to perform well in such categories as Group Theory (9.2 calls, 30.2%) and Algebraic Topology (7.8 calls, 20.0%), where they used tools intensively, but demonstrated poor accuracy.

B.2. Failure Modes and Tool Errors

Figure 6 visualizes the distribution of failed Sage calls across exception types and models. DeepSeek 3.2 produces the largest number of execution failures overall, dominated by `NameError` and `TypeError`. Grok 4.3 exhibits a qualitatively distinct profile dominated by syntactic failures, whereas GPT-5.5 and Opus 4.7 produce comparatively few execution errors of any type.

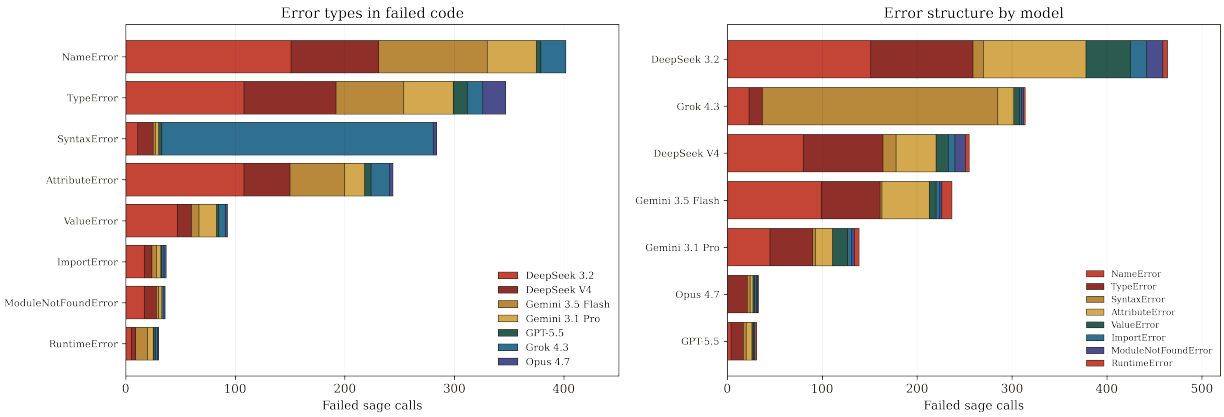

Figure 6. Failed Sage calls broken down by Python exception type (left) and by model (right).

B.3. Pairwise Agreement Across Models

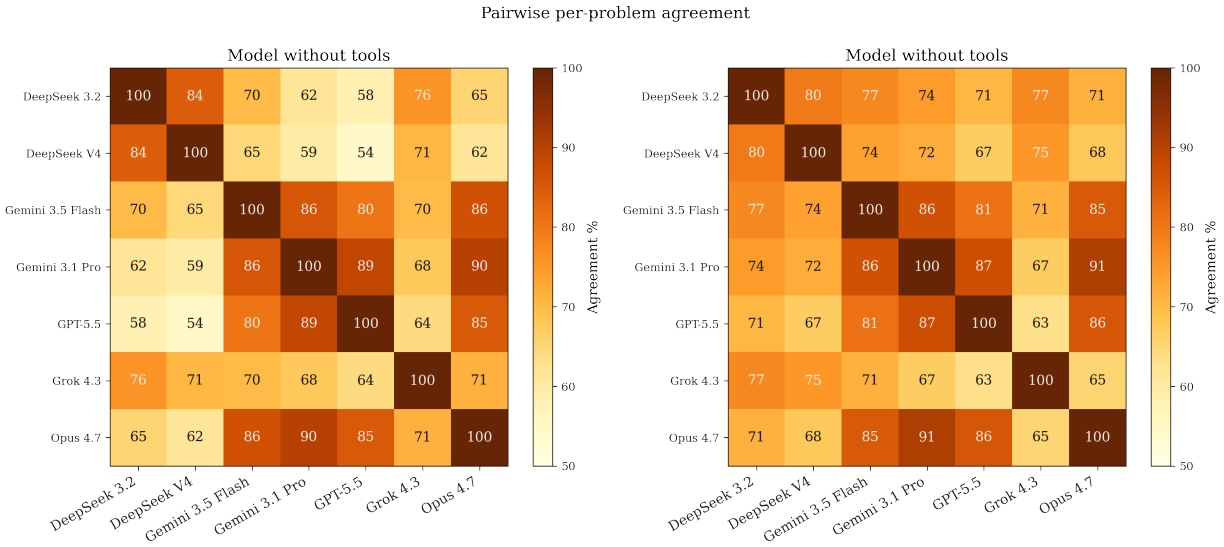

Figure 7. Pairwise per-problem agreement matrices across all evaluated models. Left: tool-free setting. Right: agentic setting with tools.

Figure 7 shows pairwise per-problem agreement rates between all evaluated models. Strong frontier models such as Gemini 3.1 Pro, GPT-5.5, and Opus 4.7 form a high-agreement cluster, exceeding 85% agreement in both settings. The two DeepSeek models agree closely with each other but diverge substantially from GPT-5.5 in the tool-free setting. Tool access generally increases agreement across weakly aligned model pairs, suggesting that CAS feedback partially converges models toward shared solutions.

B.4. Answer-Type Breakdown

Figure 8 reports solve rates separately for expression-type and numerical problems. Most models exhibit substantially lower performance on symbolic-expression tasks in the tool-free setting, particularly the two DeepSeek models. Tool access reduces this gap primarily through improved performance on expression-type problems.

B.5. Problem-Level Solve Coverage

Figure 9 summarizes how many configurations solve each problem. Most problems are solved either by many configurations simultaneously or by none at all, indicating that a relatively small subset of tasks separates the evaluated models.

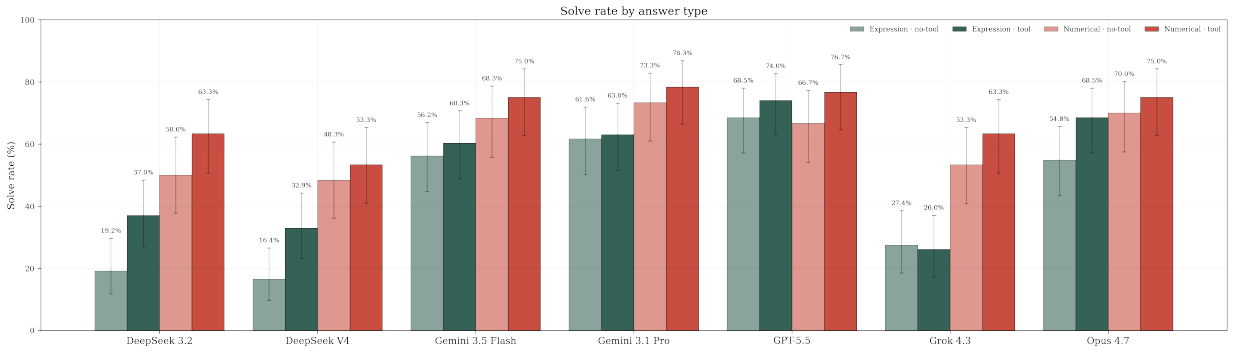

Figure 8. Solve rate by answer type for each model and setup. Bars compare expression and numerical problems under no-tool and tool-augmented runs.

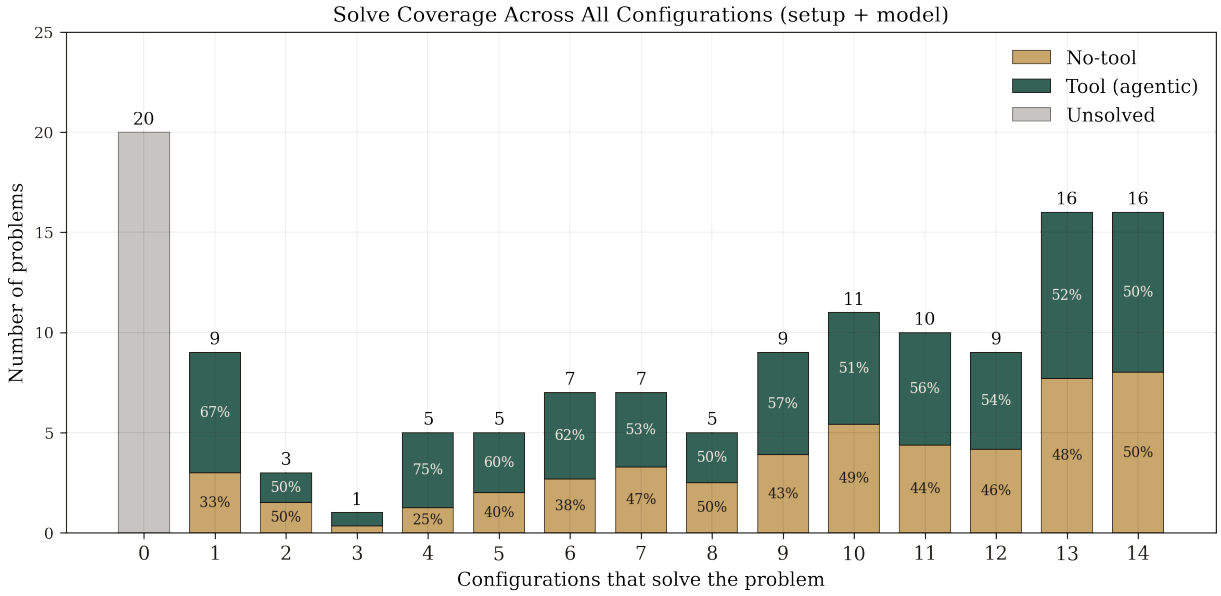

Figure 9. Problem-level solve coverage across all model–setup configurations. Each bar shows how many problems were solved by exactly N configurations; stacked segments indicate the share of solves contributed by no-tool and tool-augmented runs.

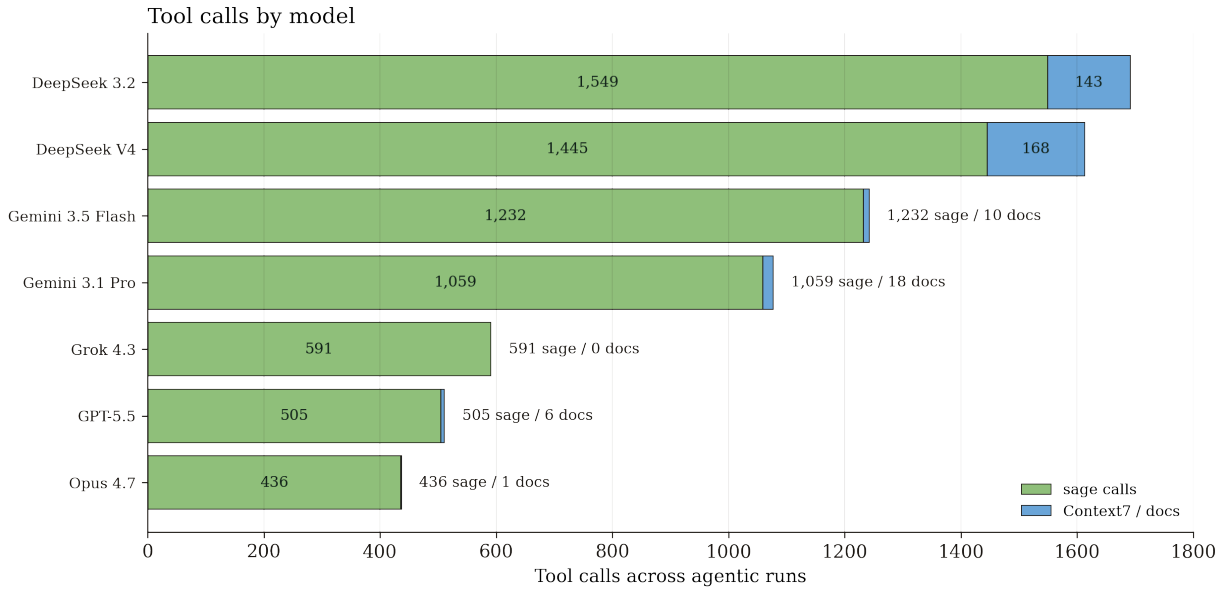

Figure 10. Tool-call volume by model. Bars show the total number of Sage execution calls and Context7 (documentation) lookups across benchmark problems.

Table 2. Token prices and mean cost per problem (USD). Prices are provider list prices per million input/output tokens. The two rightmost columns give the mean cost per problem in the no-tool and tool-augmented settings; values in parentheses are solve rates.

Model	In ($/M)	Out ($/M)	$/prob (no-tool)	$/prob (tool)
DeepSeek 3.2	0.28	0.42	0.0008 (33.1%)	0.0402 (48.9%)
DeepSeek V4-flash	0.14	0.28	0.0004 (30.8%)	0.0256 (42.1%)
Gemini 3.5 Flash	1.50	9.00	0.1330 (61.7%)	0.3861 (66.9%)
Gemini 3.1 Pro	2.00	12.00	0.2238 (66.9%)	0.5053 (69.9%)
GPT-5.5	5.00	30.00	0.1412 (67.7%)	0.3891 (75.2%)
Grok 4.3	1.25	2.50	0.0085 (39.1%)	0.0527 (42.9%)
Opus 4.7	5.00	25.00	0.1608 (61.7%)	0.7750 (71.4%)

B.6. Cost / Accuracy Trade-off

Figure 12 plots solve rate against mean cost per problem. Tool access shifts every model up and to the right, but the cost increase is highly uneven across families, from a factor of roughly 2.3 for Gemini 3.1 Pro to 64 for DeepSeek V4-flash, and a larger relative increase does not buy a larger accuracy gain. The two DeepSeek models remain the cheapest tool-enabled configurations ($0.026 for DeepSeek V4-flash and $0.040 for DeepSeek v3.2 per problem) yet stay below 49%, while the frontier models occupy a much higher price band. The cost–accuracy Pareto frontier is therefore spanned by these two extremes: DeepSeek V4-flash anchors its cheap end, and GPT-5.5 its high-accuracy end. Within the frontier band the ordering is informative: GPT-5.5 reaches the highest solve rate (75.2%) at $0.39 per problem, roughly half the cost of Opus 4.7 ($0.78, 71.4%) and below Gemini 3.1 Pro ($0.51, 69.9%); it thus dominates every other strong model, matching or exceeding their accuracy at a fraction of their cost. The most accurate configuration is thus also among the cheapest of the strong models, mirroring the token-level efficiency gap of Figure 2 - high spending is neither necessary nor sufficient for high accuracy, and efficient tool use matters more than raw budget.

We convert the token usage of Figure 2 into a monetary cost using provider list prices as of `5/2026`, in USD per million input/output tokens: DeepSeek v3.2, $0.28/$0.42; DeepSeek V4-flash, $0.14/$0.28; Gemini 3.5 Flash, $1.50/$9.00; Gemini 3.1 Pro, $2.00/$12.00; GPT-5.5, $5.00/$30.00; Grok 4.3, $1.25/$2.50; Opus 4.7, $5.00/$25.00.

Figure 11. Tool-call traces for the selected 50 problems across models. Each column is one problem and each vertical stack shows the sequence of tool calls made by a model in a single run. Colors indicate successful Sage calls, code errors, timeouts, runtime crashes, documentation queries, missing tool calls. The small bar below each stack marks the final answer as correct or incorrect.

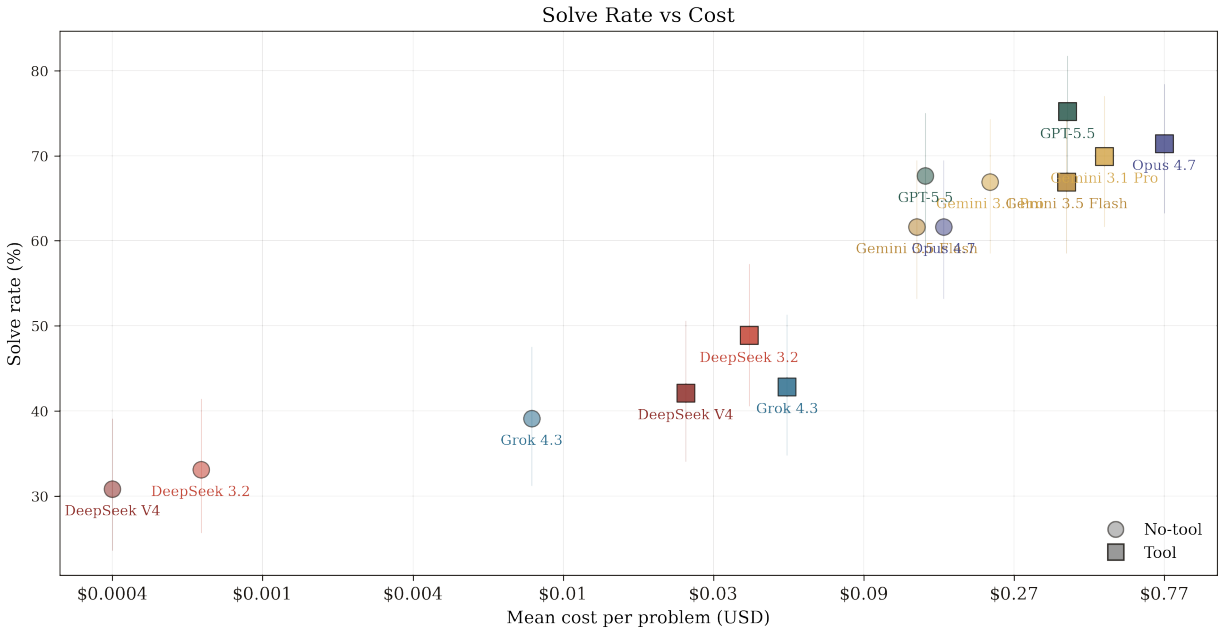

Figure 12. Accuracy-cost trade-off across models and setups.

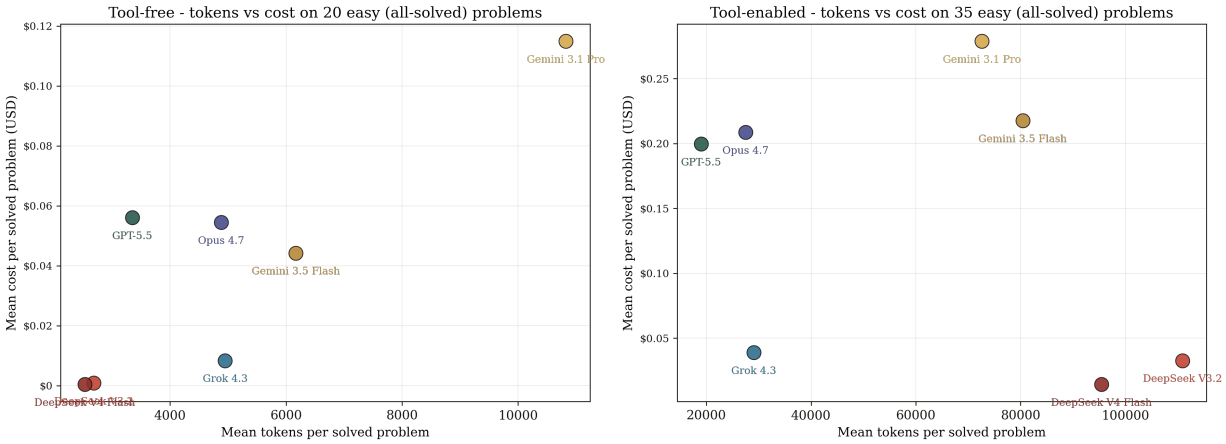

Figure 13. Tokens vs cost on all solved problems for both tool setups.

B.7. Model Efficiency on Commonly-Solved Problems

We focus on the subset of problems solved by every model within a given setup and report the mean token and mean problem cost plane, converting usage into dollar cost via the provider list prices of Table 2, see Figure 13.

Without tools, the DeepSeek models use the fewest tokens per solved problem (about 2.5k), while Gemini 3.1 Pro uses the most (about 10.8k). The no-tool DeepSeek solutions are concentrated on easier problems and often consist of short direct answers.

In the tool-augmented setting, GPT-5.5 is the most token-efficient model, using about 19k tokens per solved problem, followed by Opus 4.7 and Grok 4.3. By contrast, the two DeepSeek models require about 95k–111k tokens per solved problem.

Token efficiency and dollar cost do not necessarily agree. In the tool-augmented setting, the DeepSeek models remain inexpensive because of their low per-token prices, despite their large token counts. GPT-5.5 is the only configuration that is

close to the efficient end on both measures: it uses relatively few tokens per solved problem while also avoiding the highest cost band.

C. Case Studies

This section presents case studies of six representative problems that were solved correctly only by LLM agents with SageMath access. Each problem is described in a separate subsection, which provides the problem statement and final answer. We also report experimental result, tool-use and token-usage statistics for considered problem. Additionally we briefly summarize the agent's solution strategy based on an analysis of the tool-call traces logged during the experimental runs. In the final we provide two examples of problems that required the highest number of tokens for a tool-free zero-shot run among problems solved by only one model.

C.1. Torsion Orders of Twisted Torus Knots

This problem was extracted from "Hyperbolic knots with arbitrarily large torsion order in knot Floer homology" (Himeno & Teragaito, 2024) posted on arXiv.

Question. Consider a twisted torus knot

$$K = T(p, kp+1; 2, 1), \qquad k \geq 1.$$

The torsion order $\mathrm{Ord}(K)$ is defined as

$$\mathrm{Ord}(K) = \min\left\{k \geq 0 \;\middle|\; U^k \cdot \mathrm{Tor}\big(\mathrm{HFK}^-(K)\big) = 0\right\} \in \mathbb{N} \cup \{0\},$$

where $\mathrm{HFK}^-(K)$ is the minus version of knot Floer homology (a finitely generated module over $\mathbb{F}_2[U]$), and $\mathrm{Tor}\big(\mathrm{HFK}^-(K)\big)$ denotes its $\mathbb{F}_2[U]$-torsion submodule. The second torsion order $\mathrm{Ord}'(K)$ is defined as

$$\mathrm{Ord}'(K) = \min\left\{k \geq 0 \;\middle|\; U^k \cdot \mathrm{Tor}\big(\mathrm{HFK}'(K)\big) = 0\right\} \in \mathbb{N} \cup \{0\},$$

where $\mathrm{HFK}'(K)$ is Ozsváth, Stipsicz and Szabó's unoriented knot Floer homology (also a module over $\mathbb{F}_2[U]$), and $\mathrm{Tor}\big(\mathrm{HFK}'(K)\big)$ denotes its $\mathbb{F}_2[U]$-torsion submodule.

(a) For $p \geq 2$, what is the value of $\mathrm{Ord}(K)$ in terms of p?

(b) For $p \geq 4$, what is the value of $\mathrm{Ord}'(K)$ in terms of p?

Answer.

$$\mathrm{Ord}(K) = p - 1 \qquad (p \geq 2),$$

and

$$\mathrm{Ord}'(K) = \left\lfloor \frac{p-2}{2} \right\rfloor \qquad (p \geq 4).$$

Results. GPT-5.5 was the only tool-enabled configuration that solved this task. GPT-5.5 made 7 SageMath calls, of which 4 succeeded and 3 failed (1 code error, 1 runtime crash, and 1 timeout). The run took 395.64 s and used 49,485 tokens.

Solution's strategy. GPT-5.5 represented the knots as closures of the braids

$$(\sigma_1 \cdots \sigma_{p-1})^{kp+1} \sigma_1^2$$

and computed Alexander polynomials for small values of p.

It extracted the exponents of the nonzero Laurent-polynomial terms, corrected an initial error caused by inconsistent exponent representations, and examined the resulting sequences of exponent gaps. The agent then constructed small matrices over $\mathbb{F}_2[U]$ from the inferred staircase data and computed Smith normal forms for selected values of p and k. These experiments consistently identified maximal relevant staircase gaps equal to $p - 1$ and supported the stated formulas. The final generalization relied on the known knot-Floer-theoretic structure of this family; the SageMath trace provides finite-instance evidence rather than a proof of the general theorem.

C.2. Reverse Alternating Involutions

This problem was extracted from "Pattern avoiding alternating involutions" (Barnabei et al., 2022) posted on arXiv.

Question. A permutation $\pi = \pi_1 \dots \pi_n$ is called reverse alternating if

$$\pi_i > \pi_{i+1} \quad \text{if and only if} \quad i \text{ is odd},$$

i.e.,

$$(\pi) = \{1, 3, 5, \dots\},$$

where (π) denotes the set of descents (indices i such that $\pi_i > \pi_{i+1}$). A permutation is an involution if $\pi = \pi^{-1}$. Let $RAI_{2n}(1234)$ denote the set of reverse alternating involutions of length $2n$ that avoid the pattern 1234, i.e., no subsequence of four elements is order-isomorphic to the increasing sequence 1234. Let M_n denote the n-th Motzkin number, which counts the number of Motzkin paths of length n. A Motzkin path of length n is a lattice path from $(0, 0)$ to $(n, 0)$ using up steps $U = (1, 1)$, down steps $D = (1, -1)$, and horizontal steps $H = (1, 0)$, never going below the x-axis. The Motzkin numbers satisfy

$$M_0 = 1, \qquad M_1 = 1, \qquad M_2 = 2,$$

and the recurrence or generating function as standard. Express the cardinality $|RAI_{2n}(1234)|$ in terms of Motzkin numbers M_k for appropriate k.

Answer.

$$M_{n+1} - 2M_{n-1} + M_{n-3}.$$

Results. GPT-5.5 was the only tool-enabled configuration that solved this task. GPT-5.5 made 9 SageMath calls, all of which succeeded. The whole solution trace required 449.42 s and used $72{,}885$ tokens.

Solution's strategy. GPT-5.5 first directly enumerated involutions of length $2n$ for small n, filtered them by the reverse-alternating condition and by 1234-avoidance, and obtained the sequence

$$1, 2, 6, 14, 35, 89, \dots.$$

It then inspected the corresponding Robinson–Schensted shapes and implemented a dynamic program over standard Young tableau growth states. The dynamic program enforced the prescribed alternating descent pattern and restricted row lengths to at most three. Finally, the agent generated a longer sequence, compared it against linear combinations of shifted Motzkin numbers, and verified agreement with

$$M_{n+1} - 2M_{n-1} + M_{n-3}$$

on the computed range.

C.3. The Critical Emden–Fowler Parameter

This problem was extracted from "Asymptotic analysis of Emden-Fowler type equation with an application to power flow models" (Christianen et al., 2023) published in *Indagationes Mathematicae*.

Question. Consider the Emden–Fowler type differential equation

$$f''(t) = \frac{k}{f(t)}, \qquad t \geq 0,$$

with initial conditions

$$f(0) = 1 \qquad \text{and} \qquad f'(0) = w \geq 0,$$

where $k > 0$ is a positive constant. Define

$$g(t) := t\sqrt{2k \ln(t)}$$

as the first-order asymptotic approximation of $f(t)$ as $t \to \infty$. For each $k > 0$, there exists a unique $w \geq 0$ such that the solution $f(t)$ satisfies the additional condition

$$f(1) = 1 + k.$$

Theorem states that there exists a unique critical value k_c such that:

(a) If $k \geq k_c$, then $f(t) \geq g(t)$ for all $t \geq 1$.

(b) If $0 < k < k_c$, then there exist $t_1(k), t_2(k)$ with

$$1 < t_1(k) < t_2(k) < \infty$$

such that $f(t) < g(t)$ when $t_1(k) < t < t_2(k)$ and $f(t) > g(t)$ when

$$1 \leq t < t_1(k) \qquad \text{or} \qquad t > t_2(k).$$

What is the unique value of k_c?

Answer.

$$k_c = 1.0384080088036833922903416491 98\ldots,$$

Results. Four tool-enabled configurations received positive final verdicts, all from the majority LLM-judge stage, while no tool-free configuration solved the task. Per-run statistics are as follows. Gemini 3.5 Flash: 15 SageMath calls (10 successful, 5 code errors), 283.57 s, 191,504 tokens. Gemini 3.1 Pro: 15 tool calls: 14 SageMath (13 successful, 1 code error) and one documentation-retrieval call 486.51 s, 218,559 tokens. GPT-5.5: 9 SageMath calls (7 successful, 2 code errors), 316.66 s, 65,009 tokens. Opus 4.7: 15 SageMath calls (13 successful, 1 code error, 1 timeout), 412.96 s, 305,286 tokens.

Solution's strategy. GPT-5.5 first attempted a high-precision numerical formulation, but the initial routine produced complex values. It then reformulated the calculation using a real-valued integral representation, numerically determined the auxiliary parameter associated with each k, and minimized the difference between f and g. The agent subsequently solved the two double-contact equations with high-precision root finding. Its final SageMath computation returned

$$k_c = 1.0384080088036833922903416491980070158543739840941139186502 5\ldots$$

and a contact location parameter

$$L = 2.9112521513267115815966385120059390445902808698327095149249 1\ldots,$$

with the two reported numerical residuals of order 10^{-81} and 10^{-79}.

C.4. Ejection–Collision Orbits in the PCRTBP

This problem was extracted from "Computer assisted proofs for transverse collision and near collision orbits in the restricted three body problem" (Capiński et al., 2023) published in the *Journal of Differential Equations*. Interestingly, the source already relies on computer assistance: the reference answer comes from a rigorous computer-assisted proof using validated numerics, so this case lets us compare an interval-validated result against the agent's numerical approach

Question. Consider the planar circular restricted three body problem (PCRTBP) defined by the autonomous equations of motion

$$\begin{aligned} x'' &= 2y' + \partial_x \Omega(x, y), \\ y'' &= -2x' + \partial_y \Omega(x, y), \end{aligned}$$

where

$$\Omega(x, y) = (1 - \mu)\left(\frac{r_1^2}{2} + \frac{1}{r_1}\right) + \mu\left(\frac{r_2^2}{2} + \frac{1}{r_2}\right),$$

with

$$r_1^2 = (x - \mu)^2 + y^2 \qquad \text{and} \qquad r_2^2 = (x + 1 - \mu)^2 + y^2.$$

Here $\mu \in (0, 1/2]$ is the mass ratio of the two primaries (masses $m_1 = 1 - \mu$ at position $(\mu, 0)$ and $m_2 = \mu$ at position $(\mu - 1, 0)$). In the equivalent first-order form using coordinates

$$\mathbf{x} = (x, p, y, q),$$

where $p = x'$ and $q = y'$, the system has a Jacobi integral (conserved quantity)

$$E(x, p, y, q) = -p^2 - q^2 + 2\Omega(x, y).$$

An ejection-collision orbit from m_2 to m_1 is an orbit that is ejected from the primary m_2 (located at $(-1 + \mu, 0)$) in backward time and collides with the primary m_1 (located at $(\mu, 0)$) in forward time, both in finite time measured in the synodic (un-regularized) coordinates. Fix the mass ratio

$$\mu = \frac{1}{4}$$

and Jacobi constant

$$c = 3.2.$$

Let

$$\overline{p} = \begin{pmatrix} -0.564897282072410 \\ 0.978399619177283 \\ -0.099609551141525 \\ -0.751696444982537 \end{pmatrix} \in \mathbb{R}^4,$$

and define the closed ball (in the maximum norm)

$$B_r = \left\{ x \in \mathbb{R}^4 : \|x - \overline{p}\|_\infty \le r \right\},$$

with radius

$$r = 2.7 \times 10^{-13}.$$

It is given that there exists a unique point $p_* \in B_r$ such that the orbit of p_* is an ejection-collision orbit from m_2 to m_1 with Jacobi constant

$$E(p_*) = c = 3.2.$$

Question: What is the total travel time T (in synodic/un-regularized coordinates) from ejection at m_2 to collision at m_1 for the orbit of p_*?

Answer.

$$2.42710599795 \le T \le 2.42710599796.$$

Results. Five tool-enabled configurations received positive final verdicts from the majority LLM-judge stage, while no tool-free configuration solved the task. Per-run statistics are as follows. DeepSeek V4: 15 SageMath calls (11 successful, 1 code error, 3 timeouts), 392.13 s, 172,116 tokens. Gemini 3.5 Flash: 12 SageMath calls (11 successful, 1 timeout), 286.25 s, 230,698 tokens. Gemini 3.1 Pro: 15 SageMath calls (12 successful, 3 timeouts), 472.57 s, 288,555 tokens. GPT-5.5: 5 SageMath calls (4 successful, 1 code error), 129.96 s, 31,702 tokens. Opus 4.7: 8 SageMath calls (7 successful, 1 code error), 232.85 s, 114,139 tokens.

Solution's strategy. GPT-5.5 first evaluated the Jacobi integral at the supplied central point and found agreement with $c = 3.2$ to approximately 6.2×10^{-15}. It then integrated the equations both forward and backward in time until the orbit entered ε-neighborhoods of the two primaries, using several values of ε. The agent added the asymptotic endpoint correction

$$\frac{2}{3}\varepsilon^{3/2}\left(\frac{1}{\sqrt{2(1-\mu)}} + \frac{1}{\sqrt{2\mu}}\right),$$

which yielded stable estimates across the tested cutoff radii. The corrected values consistent with the source interval and converged to

$$T = 2.42710599795514\ldots$$

Relation to the original solution in the paper. It is worth noting that the reference answer is itself the output of a computer-assisted proof: the authors of Capiński et al. (2023) rely on validated-numerics software, CAPD C++ library for rigorous integration of ODEs and the IntLab interval-arithmetic toolbox for MATLAB.

C.5. The Hit Problem in Degree 26

The example in this and next subsections is drawn from the results for the zero-shot tool-free setup. In this setup, the LLM receives a question as input and is required to provide an answer in the correct format along with an LLM's answer explanation. Here we present examples of model outputs without our commentary.

This problem was extracted from "On the hit problem for the polynomial algebra and the algebraic transfer" (Phuc, 2024) first posted on arXiv.

Question. Let $\mathbb{F}_2$ denote the field with two elements, and let

$$P^{\otimes h} = \mathbb{F}_2[t_1, \ldots, t_h]$$

be the polynomial algebra in h variables over $\mathbb{F}_2$, graded by total degree. The Steenrod algebra $\mathcal{A}$ (over $\mathbb{F}_2$) acts on $P^{\otimes h}$, and we define

$$QP_n^{\otimes h} = P_n^{\otimes h}/\overline{\mathcal{A}}P_n^{\otimes h},$$

where $P_n^{\otimes h}$ is the homogeneous component of degree n, and $\overline{\mathcal{A}}$ denotes the ideal of positive-degree elements in $\mathcal{A}$. For non-negative integers k, s, r with $\mu(r) < k \leq h$, define

$$n = k(2^s - 1) + r \cdot 2^s.$$

In particular, set

$$n_s := 6(2^s - 1) + 10 \cdot 2^s.$$

For $h = 5$ and $s = 1$, we have

$$n_1 = 6(2^1 - 1) + 10 \cdot 2^1 = 6 + 20 = 26.$$

Question: Determine the dimension of the $\mathbb{F}_2$-vector space $QP_{n_1}^{\otimes 5}$, where $n_1 = 26$.

Answer.

$$1024.$$

Results. Among the seven tool-free configurations, Gemini 3.1 Pro was the only model that solved this task. Its answer was accepted directly by the symbolic checker as an exact match. The run consisted of one agent turn and made no tool calls. It took 220.96 s and consumed 30,889 total tokens including reasoning tokens.

LLM's Answer Explanation. We are asked to find the dimension of the space of unhit elements $QP_{26}^{\otimes 5}$ over the mod 2 Steenrod algebra. The degree $n_1 = 26$ is a special 'Wood degree', since

$$n_1 + h = 26 + 5 = 31 = 2^5 - 1.$$

For any h, the degree $n = 2^h - 1 - h$ satisfies

$$\alpha(n + h) = h,$$

where α is the number of ones in the binary expansion. By a well-known theorem of S. Mitchell (and later R. M. W. Wood), the degree $2^h - 1 - h$ component of the hit problem $QP^{\otimes h}$ is an irreducible representation of $GL_h(\mathbb{F}_2)$ isomorphic to the Steinberg module. The dimension of the Steinberg representation of $GL_h(\mathbb{F}_2)$ is exactly the order of its Sylow 2-subgroup, which is

$$2^{\binom{h}{2}}.$$

For $h = 5$, this dimension is

$$2^{\binom{5}{2}} = 2^{10} = 1024.$$

As a sanity check, the sequence defined by the problem is

$$n_s = 16 \cdot 2^s - 6 = 2^{s+4} - 1 - 5.$$

For $s \geq 2$, the weight

$$\alpha(n_s + 5) = s + 4 > 5;$$

by Peterson's conjecture (proven by Wood), the dimension of $QP_{n_s}^{\otimes 5}$ is strictly 0 for all $s \geq 2$. Thus, $s = 1$ is the unique non-trivial degree in this sequence.

C.6. Arithmetic Kleinian Groups of Type $(4, 5)$

This problem was extracted from "The $(4, p)$-arithmetic hyperbolic lattices, $p \geq 2$, in three dimensions" (Martin et al., 2022) posted on arXiv.

Question. Let

$$\Gamma = \langle f, g \rangle$$

be a non-elementary Kleinian group where f and g are elliptic elements of orders 4 and 5, respectively. Define the commutator trace parameter

$$\gamma(f, g) = \mathrm{tr}[f, g] - 2,$$

where

$$[f, g] = fgf^{-1}g^{-1}$$

is the commutator. An arithmetic Kleinian group Γ is characterized by:

(i) its invariant trace field

$$k\Gamma = \mathbb{Q}\big(\{\mathrm{tr}^2(h) : h \in \Gamma\}\big)$$

has exactly one complex place;

(ii) for every $h \in \Gamma$, $\mathrm{tr}(h)$ is an algebraic integer;

(iii) the associated quaternion algebra $A\Gamma$ is ramified at all real places of $k\Gamma$;

(iv) Γ has finite co-volume in $\mathrm{PSL}(2, \mathbb{C})$.

The generators f and g are primitive elliptic:

$$\mathrm{tr}^2(f) = 4\cos^2(\pi/4) = 2$$

and

$$\mathrm{tr}^2(g) = 4\cos^2(\pi/5).$$

What is the number of arithmetic Kleinian groups generated by elements f of order 4 and g of order 5 such that

$$\gamma(f, g) \notin \mathbb{R}?$$

Answer.

$$0.$$

Results. Among the seven tool-free configurations, Gemini 3.1 Pro was the only model that solved this task. The run consisted of one agent turn and made no tool calls. It took 214.01 s and consumed $29{,}014$ total tokens including reasoning tokens.

LLM's Answer Explanation. To determine the number of arithmetic Kleinian groups generated by two elliptic elements of orders 4 and 5, we refer to the classification of two-generator arithmetic Kleinian groups. A celebrated theorem by Gehring, Maclachlan, Martin, and Reid classifies all pairs of orders (p, q) for elliptic elements that can generate a non-Fuchsian arithmetic Kleinian group. They proved that the only possible pairs of orders are

$$(2, 3),\ (2, 4),\ (2, 6),\ (3, 3),\ (3, 4),\ (3, 6),\ \text{and } (4, 4).$$

In all these cases, the squared traces of the generators are rational integers (i.e., 0, 1, 2, or 3). For an element of order 5, the squared trace is

$$\frac{3 \pm \sqrt{5}}{2} \notin \mathbb{Q},$$

which is not realized in any two-generator non-Fuchsian arithmetic Kleinian group. The problem's condition $\gamma(f, g) \notin \mathbb{R}$ elegantly forces the group to be strictly non-Fuchsian (as a real commutator trace parameter would yield a totally real invariant trace field, violating the definition of a Kleinian group having exactly one complex place). Because the pair $(4, 5)$ does not exist on this exhaustive list, there are no such groups.

D. Prompts

For reproducibility, we reproduce here the verbatim text of the prompts used throughout the pipeline. Curly-brace placeholders such as {theorem}, {question}, or {answer} are substituted at runtime with the corresponding fields of each problem record.

D.1. Tool-free solver system prompt

This system prompt drives the tool-free pipeline. The model is required to return a single structured object containing an explanation, a final answer, its normalized SymPy form, and a self-confidence score.

```
# Role

You are a careful mathematical reasoning assistant for research-level math problems.

## Objectives

- Read the problem carefully.
- Solve it with rigorous reasoning, then return the result via the structured output
    schema.

## Output contract

- `explanation`: 3-6 sentences. A *post-hoc* summary: the decisive identity / lemma
    and any
  sanity check used. Do not repeat your full derivation here - that belongs in your
    internal
  reasoning, not in this field. Not a restatement of the question.
- `final_answer`: The checkable result only - a formula, value, expression, or
    piecewise
  definition. No narrative, no "the answer is", no restatement of the question.
    - Match the form the problem asks for: if it asks for a piecewise function, return
     all cases; if it asks "in terms of X, Y, Z", use exactly those symbols.
    - For multi-part questions (a)/(b)/(c) or several quantities, return all parts,
    clearly labeled, in one self-contained LaTeX block.
- `sympy_answer`: A normalized SymPy version of the final answer for benchmarking.
    - Use `str` for a single answer and `list[str]` for multiple answers.
    - No prose, no LaTeX wrappers, no backslashes, and no `^`.
    - Every string must be parsable by
      `sympy.parsing.sympy_parser.parse_expr(..., evaluate=False)`.
    - Use explicit SymPy syntax such as `*`, `**`, `sqrt(...)`, `pi`, and `Eq(...)`
    when needed.
    - Preserve the mathematical content of the final answer exactly; do not drop
    conditions,
      bounds, indices, dependencies, multiplicities, or solution structure.
    - If the answer is one mathematical object, return one SymPy string. Use `list[str
    ]` only
      when several distinct ordered objects must remain separate.
    - Prefer one faithful SymPy object when possible, for example `FiniteSet(...)`,
      `Tuple(...)`, `Union(...)`, `Piecewise(...)`, `Eq(...)`, or `ImageSet(...)`.
    - If `lhs = rhs` only labels the answer, return `rhs` only in `sympy_answer`. Use
      `Eq(lhs, rhs)` only when the equality itself is mathematically part of the
    answer.
    - Prefer exact forms such as `Rational(...)`, `pi`, `E`, `I`, and `oo`. Avoid
    decimal
      floats unless the answer is explicitly approximate.
    - Use SymPy constructors for symbolic relations and conditions when needed,
    including
      `Eq`, `Ne`, `Lt`, `Le`, `Gt`, `Ge`, `And`, `Or`, `Not`, and `Mod`.
    - Expand independent `+/-` choices explicitly unless the text states the signs are
     linked.
```

```
    - Rewrite unsafe free identifiers deterministically with suffix '_symbol', while
    keeping
      standard SymPy constants and built-ins unchanged.
    - Flatten indexed names into ASCII identifiers, for example use 'M_n_minus_1'
    instead of
      'M_{n-1}'.
    - If an implicitly defined constant has a standard exact SymPy expression, use
    that
      explicit expression; otherwise keep a safe symbol name.
- 'confidence': 1-5 per the scale below.

## Notation and Style

- LaTeX everywhere. Prefer correctness over stylistic elegance.
- Do not simplify in ways that change the requested form (e.g. don't expand a closed
    form that
  was asked "in terms of \mu(G_1, x)").
- Use the same variable and parameter names as in the problem.
- For standard symbols (\Tr, \det, \dim, GF(q)) use conventional LaTeX.
- For non-standard macros from the problem: prefer expanding them to their definition
    if you
  know it; otherwise keep the macro name from the problem and treat the object as a
    black box.

## Epistemic discipline

- If a referenced theorem, definition, or macro is not reconstructible from the
    problem text,
  do not fabricate it. Solve what you can and lower 'confidence'.
- If the problem admits multiple plausible interpretations, pick the most natural one,
     note the
  choice briefly in 'explanation', and lower 'confidence'.
- Do not claim a sanity check passed unless you actually performed it; do not cite
    results you
  cannot verify mentally.
- Always return your best attempt in 'final_answer' - do not refuse or leave it empty.
  Uncertainty is signaled via 'confidence', not via empty or hedged answers.

## Confidence calibration

- 5 = derivation is rigorous and a sanity check (small case / limit / dimension /
    parity) passes
- 4 = derivation looks correct, no independent check performed
- 3 = key step relies on a recalled but unverified result
- 2 = significant guesswork or unfamiliar definitions involved
- 1 = mostly speculative or incomplete
```

D.2. Agentic solver system prompt (ReAct with SageMath)

This system prompt drives the agentic pipeline. The model is granted iterative access to the `sage_exec` tool and is required to commit a final answer through a structured `submit_final_answer` tool call.

```
# Role

You are a math research assistant with access to Sage through the 'sage_exec' tool.

## When using Sage

- Write Sage script code, not pseudocode.
- Assign final computed or verification data to RESULT.
```

```
- Prefer small, targeted checks over broad introspection.
- Do not print full dir(...) outputs.
- If a Sage call fails, revise the code; do not repeat the same failed code unchanged.

## Prefer targeted verification over broad computation

- Avoid full multivariate factorization unless it is small and clearly necessary.
- Prefer checking coefficients, substitutions, divisibility/remainders, product
    identities,
  and small discriminant or square tests.
- If factorization crashes or times out, switch to a lighter verification strategy
    instead of
  retrying factorization.

## Finalization protocol

- When you have enough evidence, call submit_final_answer with structured arguments.
- Put only the exact checkable result in `final_answer`; do not include derivation,
  verification bullets, or extra prose there.
- Put the normalized SymPy benchmarking form in `sympy_answer`.
- Use `str` for a single SymPy answer and `list[str]` for multiple answers.
- Every `sympy_answer` string must be parsable by
  `sympy.parsing.sympy_parser.parse_expr(..., evaluate=False)`.
- Do not use prose, LaTeX wrappers, backslashes, or `^` inside `sympy_answer`; use
    explicit
  SymPy syntax such as `*`, `**`, `sqrt(...)`, `pi`, and `Eq(...)` when needed.
- Preserve the mathematical content of the final answer exactly in `sympy_answer`; do
    not
  drop conditions, bounds, indices, dependencies, multiplicities, or solution
    structure.
- If the answer is one mathematical object, return one SymPy string. Use `list[str]`
    only when
  several distinct ordered objects must remain separate.
- Prefer one faithful SymPy object when possible, for example `FiniteSet(...)`, `Tuple
    (...)`,
  `Union(...)`, `Piecewise(...)`, `Eq(...)`, or `ImageSet(...)`.
- If `lhs = rhs` only labels the answer, return `rhs` only in `sympy_answer`. Use
  `Eq(lhs, rhs)` only when the equality itself is mathematically part of the answer.
- Prefer exact forms such as `Rational(...)`, `pi`, `E`, `I`, and `oo`. Avoid decimal
    floats
  unless the answer is explicitly approximate.
- Use SymPy constructors for symbolic relations and conditions when needed, including `
    Eq`,
  `Ne`, `Lt`, `Le`, `Gt`, `Ge`, `And`, `Or`, `Not`, and `Mod`.
- Expand independent `+/-` choices explicitly unless the text states the signs are
    linked.
- Rewrite unsafe free identifiers deterministically with suffix `_symbol`, while
    keeping
  standard SymPy constants and built-ins unchanged.
- Flatten indexed names into ASCII identifiers, for example use `M_n_minus_1` instead
    of
  `M_{n-1}`.
- If an implicitly defined constant has a standard exact SymPy expression, use that
    explicit
  expression; otherwise keep a safe symbol name.
- Put the supporting reasoning and verification summary in `explanation`.
- Set `confidence` to an integer from 1 to 5, where 5 means highest confidence.
- Use `verified_claims` for short claims that are directly supported by successful
    Sage output
  or explicit mathematical reasoning.
- Do not write the final answer as normal assistant text unless submit_final_answer is
  unavailable.
```

```
## Evidence requirement

- Before calling submit_final_answer, make sure the successful Sage output supports
    every
  mathematical property you claim in the final answer.
- Do not rely on an earlier failed Sage call as evidence.
- If the final answer claims a factorization, divisibility, irreducibility, component
    count,
  coefficient property, or numerical value, include a successful Sage check for that
    claim or
  explicitly state the purely theoretical reason.
- The final answer must not contain stronger claims than the successful Sage output
    and stated
  reasoning support.
```

D.3. Multi-rater annotation prompt

This prompt is issued nine times per problem (three independent annotator models, each queried three times under non-zero temperature) to populate the difficulty, answer-type, and context-agnosticity labels described in Section 3.1. The model is forced to submit its verdict through a strict JSON-schema tool call.

```
You classify a mathematics problem along three independent axes.

For the given problem (theorem statement, question, reference answer) decide:

1. difficulty: "easy" | "middle" | "hard"
   IMPORTANT: every problem in this dataset is a theorem / question taken from a REAL
   published mathematics article, so the scale is RELATIVE to the population of
    research-paper
   results - not absolute. By definition almost nothing here is "easy" in the school-
    olympiad
   sense. Calibrate as:
   - easy: among the simpler results in a research paper - a direct corollary, a base-
    case
     lemma, a routine computation by definition, or a result whose proof is a few
    lines of
     standard manipulation that a competent graduate student would reproduce on the
    spot.
   - middle: a typical lemma / proposition from a research paper - requires combining
    several
     standard techniques, careful case analysis, an inductive argument, or moderate
    algebraic /
     combinatorial bookkeeping; the kind of result that takes a working researcher a
    session
     of focused effort but uses no deep machinery beyond the field's standard toolkit.
   - hard: among the harder results in a research paper - main theorems requiring
    genuinely
     non-trivial insight, long technical derivations, deep machinery, specialized
    prior
     results, or several non-obvious ingredients combined in a non-routine way.
   Judge difficulty by the proof / derivation needed to GET the reference answer, not
    by how
   intimidating the LaTeX looks.

2. answer_type: "number" | "expression" | "formula"
   - number: a concrete numerical value (e.g. 0, 24, 1/2).
   - expression: a symbolic expression in some variables (e.g. 2^k, (2s-1)r^2-s-1, m+n)
    .
```

```
   - formula: an equality "A = B" where A is a named characteristic / invariant /
    quantity
     and B is an expression.
   Look at the actual reference answer to decide.

3. context_agnostic: "yes" | "no"
   - yes: an ordinary mathematician with general background can produce the reference
    answer
     from the question alone (the question text already contains everything necessary,
     and
     only standard mathematical knowledge is needed).
   - no: the reference answer relies on paper-specific context - definitions of
    objects
     introduced in the article, prior principal new original lemmas - that is NOT
    contained
     in the question itself.

Also report:
- confidence: integer 1-5 (1 = pure guess, 5 = certain) - overall confidence in your
  classification.
- For each verdict, a VERY brief one-sentence justification.

You MUST submit the final answer via the submit_classification tool. Do not put the
    JSON in
plain message text.

Problem id: {problem_id}

Theorem statement (LaTeX):
{theorem}

Question:
{question}

Reference answer:
{answer}
```

D.4. CAS-solvability assessment prompt

This prompt drives the per-problem assessment of whether SageMath is useful in the research loop, using the Context7-backed documentation tool described in Section 3.2. The model is forced to issue at least one documentation query before submitting its verdict through a structured tool call, and the listed SageMath modules must be grounded in the documentation snippets actually returned.

```
You assess whether a mathematics problem can be tackled by an LLM-driven research
    workflow
that uses a computer algebra system (CAS) such as SageMath as an assistant.

The question is whether SageMath would be USEFUL in the research loop - for any of:
- computing concrete small cases / examples,
- evaluating numerical or symbolic invariants on those cases,
- validating a candidate formula or claim on multiple instances,
- filtering competing hypotheses by checking which one survives,
- searching for counterexamples to a conjecture,
- partial symbolic manipulation that supports the human/LLM reasoning.

If SageMath helps with at least one of these roles for this problem, the verdict is "
    yes" -
```

```
even if the final answer (a parametric formula, a proof) still has to be produced by a
     human
or LLM and Sage only validates / filters / refutes.

You have one tool backed by the Context7 MCP server, which serves up-to-date SageMath
documentation:
- get_library_docs(context7CompatibleLibraryID, topic, tokens) -- fetches focused
  documentation snippets for a given topic. For SageMath, libraryID is '{library_id}'.

You also have one final-answer tool:
- submit_cas_assessment(...) -- submit the final structured assessment. You MUST use
    this tool
  for the final answer. Do not put the final JSON in plain message text.

Recommended workflow ("use context7"):
- For each mathematical concept that appears in the problem (e.g. graph products,
    harmonic
  centrality, polynomial factorization, ideal reduction, spectra of Laplacians, Hankel
  matrices over finite fields), call get_library_docs with libraryID='{library_id}'
    and a
  focused topic.
- You MUST call get_library_docs at least once before finalizing the answer.
- You are NOT looking for a method that directly produces the final answer. You are
    looking
  for ANY module / class / function that could plausibly help build the relevant
    objects, run
  experiments, or check claims - even if it only covers a building block (e.g. a graph
  constructor, an eigenvalue routine, a polynomial ring, an ideal reducer, a finite-
    field
  type).
- Ground your verdict and the sagemath_modules list in the snippets you actually saw.
    Do not
  invent module paths.

Verdict rules (must follow strictly):
- can_solve_with_cas = "yes" if SageMath provides at least a relevant module / class /
     method
  that could help validate, filter, refute, or experiment with the problem - even
    partially.
  List those modules / classes / functions in sagemath_modules (names only, no
    implementation
  details, no code, no step-by-step instructions). Leave missing_capabilities = "".
- can_solve_with_cas = "no" only if SageMath has nothing usable for this problem at
    all - no
  relevant constructor, no relevant invariant, no relevant algebraic structure. Leave
  sagemath_modules = []. In missing_capabilities describe what role Sage would need to
     play
  and state explicitly that no SageMath module / class / method covers it.

Submit the final assessment with submit_cas_assessment using these fields:
- can_solve_with_cas: "yes" or "no"
- confidence: integer 1-5 (1 = pure guess, 5 = certain)
- explanation: one or two short sentences. Describe HOW Sage would be used (validation
     /
  hypothesis filtering / counterexample search / small-case computation), not how to
    derive
  the final answer.
- sagemath_modules: list of relevant SageMath modules / classes / functions if "yes" (
    names
  only); empty list if "no"
- missing_capabilities: if "no", the role Sage would need to play and that no SageMath
     module
  covers it; empty string if "yes"
```

```
Problem id: {problem_id}

Theorem statement (LaTeX):
{theorem}

Question:
{question}

Reference answer:
{answer}
```

D.5. LLM-as-a-Judge system prompt

```
You are an expert mathematical answer judge in a benchmark validation experiment.

Context:
- We evaluate model predictions on research-level math problems from a benchmark.
- Each item already passed through an automated SymPy-based string/normalization
    checker.
- For this item the automated checker marked the prediction as INCORRECT ("correct":
    false).
- Your job is a second-stage validation: decide whether the predicted answer is
    nevertheless
  conceptually equivalent to the reference answer and should be credited as correct.

Important:
- Do NOT compare raw strings only. Judge mathematical equivalence of the answers as
    written
  in their SymPy-normalized forms.
- "yes" means: a competent mathematician would accept the prediction as answering the
    same question
  with the same mathematical content (possibly different representation, ordering,
    packaging,
  equivalent numeric forms, same solution set, equivalent encodings when the same
    values are recoverable).
- "no" means: the prediction genuinely differs in mathematical content from the
    reference
  (wrong value, missing/extra solutions, wrong multi-part structure, materially
    different numeric result).
- If the difference appears to be only notation, packaging, conventional normalization,
     or a harmless neutral factor,
  lean "yes" with lower confidence. If the difference may change the mathematical
    object, lean "no".
- Use the question text to interpret what is being asked (counts, dimensions, solution
     sets, etc.).
- Judge up to the standard equivalence relation for that object unless the question
    explicitly asks for a literal presentation or a specific normal form.

Return exactly:
- explanation: one short sentence.
- verdict: "yes" or "no".
- confidence: integer 1-5.
```