TOOLMATH: A DIAGNOSTIC BENCHMARK FOR LONG-HORIZON TOOL USE UNDER SYSTEMATIC TOOL-CATALOG CONSTRAINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce TOOLMATH, a math-grounded diagnostic benchmark for evaluating long-horizon tool use under controllable tool-catalog conditions. Its construction pipeline provides a systematic methodology for building tool-use environments: it converts stepwise solutions from the MATH dataset into reusable tools with descriptions and typed schemas, validates their behavior and usability, and applies human repair and authoring to unresolved cases. The human-authored TOOLMATH-HARD split complements the automatic pipeline by recovering unresolved cases, extending evaluation beyond successful automatic trajectories and mitigating validation-related coverage bias. Gold tools and graded distractors model connected operations, overlapping functionality, and missing capabilities. Beyond final accuracy, TOOLMATH emphasizes three evaluation axes: (1) *Adaptability* measures how much Gold-only success is retained when gold tools are replaced entirely by distractors; (2) *Robustness* measures success retention when distractors are added alongside gold tools; and (3) *Tool Connectivity* measures answer accuracy as a function of the maximum depth of verified tool-call dependencies. Behavior-conditioned metrics and trace-level failure analyses distinguish reliable tool use, tool avoidance, adaptive substitution, and failures under unreliable catalogs. Together, this construction methodology and these diagnostics provide a controlled testbed for studying how models adapt to changing tool availability, remain robust to distractors, and maintain correctness across connected tool-use trajectories.

1 INTRODUCTION

Tool-augmented language models are increasingly used as interfaces to APIs, code, and structured utilities (Schick et al., 2023; Qin et al., 2024; Patil et al., 2024; Li et al., 2023). In realistic deployments, models often face large tool catalogs assembled by retrieval or system design, where multiple tools appear relevant, distractor tools overlap semantically with intended tools, and some required capabilities may be missing (Karpas et al., 2022; Patil et al., 2025). Evaluating tool-augmented models therefore requires more than measuring final answer accuracy under a fixed tool list: we need to measure how models adapt when intended tools disappear, how robustly they behave as distractors become more similar, and whether they can maintain correctness across logically connected multi-step tool-call chains.

We introduce TOOLMATH, a math-grounded diagnostic benchmark for controlled evaluation of long-horizon tool use under systematically varied tool-catalog conditions. TOOLMATH converts human-annotated MATH solution steps into reusable Python tools with natural-language descriptions and typed schemas, and pairs each problem with a tool environment that supports sequential tool use, intermediate-output reuse, and multi-step tool composition (Hendrycks et al., 2021). Since tools are exposed only through names, descriptions, and input schemas, while implementations remain hidden, TOOLMATH directly evaluates tool selection, parameterization, and observation-conditioned reasoning rather than direct access to code. The benchmark comprises 7,699 questions and 12,369 tools, alongside a curated hard split, TOOLMATH-HARD, containing 329 challenging questions and 362 human-authored tools.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We choose mathematics as the construction domain for three reasons. First, mathematical answers enable objective automatic scoring. Second, stepwise solutions naturally induce logically connected tool-use trajectories, allowing us to evaluate whether models can compose tool calls over multiple dependent steps. Third, many problems admit alternative solution paths, which makes it possible to observe whether models can still solve some instances when gold tools are removed and only distractor tools remain.

Validation and construction methodology. Tool-wise checks, question-wise execution, and targeted human repair form a construction methodology for the catalog properties above: reusable operations must be composed through usable interfaces, related tools can partly substitute for one another, and intended capabilities may be missing. TOOLMATH-HARD supplies human-authored tools for cases unresolved by automatic construction and validation, extending coverage beyond successful automatic trajectories. This complementary route mitigates dependence on validator capabilities; Section 3.1 details the pipeline.

Controlled tool-catalog conditions. We evaluate Gold-only, Gold-present, and Distractors-only catalogs by varying intended-tool availability and distractor similarity. Detected strict equivalents are excluded, while non-equivalent tools that support alternative computations remain eligible (Section 4.1). These conditions separate long-horizon connectivity, distractor robustness, and missing-capability adaptation.

Diagnostic metrics. Tool-call rate, Tool-Acc, and NoTool-Acc characterize whether models solve by using tools, avoiding tools, or over-calling tools in unreliable catalogs. *Adaptability* measures how much Gold-only success is retained when gold tools are entirely replaced by distractors; we further distinguish retention with and without tool calls. *Robustness* measures how much Gold-only success is retained when distractors are added alongside gold tools. *Tool connectivity* evaluates accuracy as a function of the maximum depth of verified output-to-input dependencies between tool calls.

In summary, our contributions are: (i) a math-grounded benchmark of reusable hidden-code tools and long-horizon tool-composition episodes derived from MATH solutions; (ii) a tool-construction methodology combining validation, repair, and complementary human authoring to improve reliability and coverage; (iii) controlled Gold-only, Gold-present, and Distractors-only conditions with graded distractor similarity for evaluating redundancy, missing capabilities, and catalog difficulty; (iv) behavior-conditioned accuracy and diagnostics for Adaptability, Robustness, and Tool Connectivity; and (v) trace-level failure co-occurrence analyses across tool-catalog environments. Together, these components make TOOLMATH a controlled testbed for studying how language models adapt to changing tool availability, remain robust to distractors, and maintain correctness across long-horizon tool-use trajectories.

2 RELATED WORK

Prior work on tool-augmented language models spans learning to invoke tools and APIs, as well as benchmark suites that standardize function-calling formats and evaluation protocols (Schick et al., 2023; Li et al., 2023; Qin et al., 2024; Patil et al., 2024; Xu et al., 2023; Patil et al., 2025). Another line of work studies deployment-facing failures under imperfect tool access, including missing capabilities and underspecified tasks (Treviño et al., 2025). Separately, tool-use methods and agentic interfaces such as ReAct and controller-style systems focus on how models interleave reasoning with actions and orchestrate multi-step interactions (Yao et al., 2023; Shen et al., 2023). Finally, structured task generators explicitly model tool dependencies, e.g., via graph-based construction of tool-use tasks (Shen et al., 2024).

Math reasoning datasets such as GSM8K and MATH provide correctness-checkable targets and naturally decomposed multi-step solution structure (Cobbe et al., 2021; Hendrycks et al., 2021). We use MATH's annotated solution as a construction substrate: extracting reusable tool primitives from solution steps and forming long-horizon tool-composition episodes with deterministic scoring.

Overall, existing work addresses function calling, imperfect access, controllers, and dependency construction, but typically focuses on one or two axes at a time. TOOLMATH combines deterministic math-grounded scoring, reusable hidden-code tools, logical-hop annotations, graded distractor

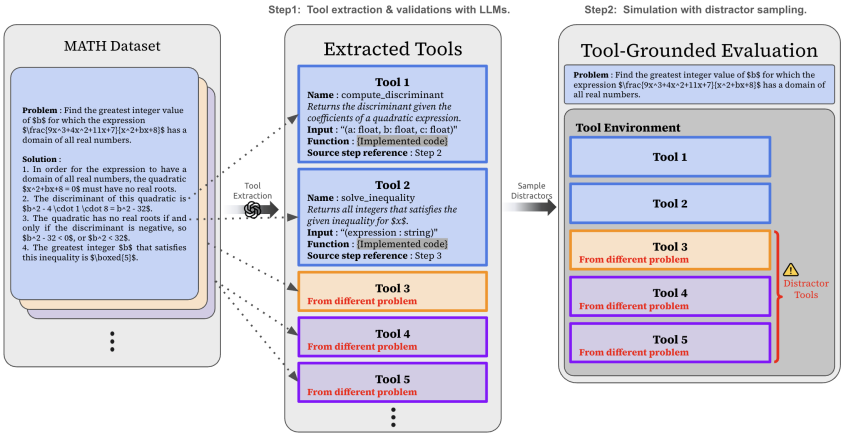


Figure 1: Construction and evaluation pipeline for TOOLMATH. **Step 1:** Extract schema-specified Python tools from MATH solutions and validate description–execution consistency. **Step 2:** Form controlled tool environments by combining each problem's gold tools with distractors sampled from other problems.

similarity, and *Distractors-only* settings to analyze sequential dependence, redundancy, and tool insufficiency jointly.

3 THE TOOLMATH DATASET

Dataset and tool definitions. An instance $(p, \mathcal{S}(p))$ pairs a MATH problem p with a tool environment (Hendrycks et al., 2021). Each tool has a name, natural-language description, typed input schema, deterministic Python implementation, and metadata linking it to source problems and solution steps. The main split contains 12,369 tools and 7,699 questions retained after construction and validation. "Gold" identifies tools intended for the reference computation; alternative solutions remain possible.

Design Objectives. TOOLMATH is designed around four objectives: (i) tools should be reusable mathematical operations rather than problem-specific macros; (ii) natural-language descriptions and typed schemas should align with executable behavior; (iii) tools should support multi-step solution plans with measurable logical-hop complexity; and (iv) distractors should be plausible rather than trivially irrelevant, enabling controlled evaluation of misleadingly similar tools, missing-capability settings, and alternative valid solution trajectories.

3.1 CONSTRUCTION PIPELINE

From MATH Solutions to Tools. Following the pipeline in Figure 1, we begin from annotated MATH solutions and decompose solution steps into reusable operations. Each tool's description is sufficient for a model to infer correct usage without seeing code. For extracted tools that are problem-specific or underspecified, we generalize them by abstraction, clarifying constraints and edge cases, and enriching descriptions to reduce ambiguity.

Validation and Final Dataset Composition. TOOLMATH applies a three-stage validation-and-repair pipeline to ensure that benchmark errors are not conflated with model errors. First, *tool-wise validation* checks description–schema–implementation consistency through schema-valid executions and multi-model judgments. Second, *question-wise validation* runs the same Plan+ReAct protocol used at evaluation time and retains episodes for which at least one gold tool is used in a correct trace. Third, ambiguous or rejected cases undergo human audit and repair, followed by re-validation. Full validator prompts, thresholds, model lists, and audit guidelines are provided in Appendix C.

3.2 TOOLMATH-HARD

TOOLMATH-HARD contains 329 questions for which automatically extracted tools did not yield a validated tool-use trajectory. Failure can reflect unsuitable extraction or a validator's unsuccessful use of an otherwise valid tool. Human annotators inspect these problems and directly construct 362 gold tools, validated separately. This complementary route mitigates validation-related coverage bias by recovering evaluation instances without requiring a prior successful trajectory using the automatically extracted tools. Human authoring is therefore part of the construction methodology, extending evaluation beyond capabilities already exposed by automatic construction and validation. The split retains the selection effect of choosing unresolved cases; it does not establish an unbiased sample of all tool-use tasks.

Questions with at least five logical hops account for 74.47% of TOOLMATH-HARD and 40.36% of the main split (Appendix L.1, Table 7). This reference-solution difference complements the behavioral results in Section 5.5; models may still use alternative solutions or answer without tools.

3.3 LOGICAL-HOP ANNOTATION

For each problem p, the logical-hop annotation $h(p)$ summarizes sequential reasoning groups in its reference solution. Alternative solution paths and no-tool answers remain possible. We distinguish $h(p)$ from the maximum verified dependency depth $D(\tau)$ of a model trajectory (Section 4.4.1).

4 EXPERIMENTAL SETUP

4.1 TOOL-SET CONSTRUCTION AND DISTRACTOR SETTINGS

For problem p, let $\mathcal{G}(p)$ be its gold tools and $\mathcal{U}$ the validated candidate pool. A distractor is non-gold for p but may support an intermediate computation or a different solution.

Excluding strict equivalents. Tools are operationally screened as strictly equivalent when their input-type multisets and output types match and their descriptions specify the same semantic operation; argument names and order alone do not distinguish functionality. Let $\widehat{\mathcal{E}}(p)$ contain detected equivalents of gold tools. Every distractor list uses $\mathcal{U}_{\setminus p} = \mathcal{U} \setminus (\mathcal{G}(p) \cup \widehat{\mathcal{E}}(p))$, before sampling, retrieval, nested lists, and fallback. All reported results use this exclusion. Appendix D.1 gives the operational screen and coverage; this is not an exhaustive proof of program equivalence.

Catalog conditions. For similarity level $\ell \in \{1, \ldots, 5\}$ and budget $k \in \{5, 10, 20, 50\}$, distractors satisfy $\mathcal{D}_{\ell,k}(p) \subseteq \mathcal{U}_{\setminus p}$. The **Gold-only condition** exposes $\mathcal{G}(p)$, the **Gold-present condition** exposes $\mathcal{G}(p) \cup \mathcal{D}_{\ell,k}(p)$, and the **Distractors-only condition** exposes only $\mathcal{D}_{\ell,k}(p)$. Gold-only is the $k = 0$ special case of Gold-present. Distractors-only removes intended tools and their detected strict equivalents, but does not rule out successful reasoning or composition of non-equivalent operations. For example, vector addition can support a complex-arithmetic computation through a change of representation without being the same typed operation as the intended complex-arithmetic tool.

Similarity levels. Levels 1–3 sample uniformly from different-category, unrestricted, and same-category eligible tools, respectively. Level 4 ranks description-embedding similarity to gold tools; Level 5 prioritizes keyword overlap with embedding similarity as a tie-breaker. Level 1 is a low-overlap sampling policy, not a guarantee of irrelevance or globally minimum similarity. Partial overlap is deliberate: models must distinguish misleading alternatives from useful substitutes.

Nested distractor guarantee and safeguards. For each (p, ℓ), we precompute 100 ordered eligible distractors and use the first k, preserving nested inclusion. Fixed seeds and serialized pool order make sampling deterministic. Unique name suffixes prevent identifier collisions separately from equivalence screening. Appendix D gives sampling and fallback rules.

4.2 MODELS

We evaluate a mix of proprietary and open models: **GPT-5**, **Claude Sonnet 4.6**, **Gemini 3.1 Pro**, **GPT-4o-mini**, **Llama 3-8B**(Grattafiori et al., 2024), and **Qwen 2.5-7B**(Yang et al., 2024a).

4.3 TOOL-USE PROTOCOLS

Unless stated otherwise, we use Plan+ReAct (Yao et al., 2023): a brief plan followed by alternating reasoning and structured tool calls (name and schema-conforming JSON arguments), conditioned on executed outputs. We log full trajectories; Appendix E provides the protocol prompts and execution rules.

4.4 EVALUATION CONDITIONS AND METRICS

We evaluate the catalog conditions, levels, and budgets defined in Section 4.1. We also report a **No tools baseline**, in which the model answers without access to any tools; this is a baseline, not a tool-catalog condition. We report exact-match answer accuracy after standard normalization. We also analyze tool invocation, tool identities, and verified output-to-input dependencies. Unless specified otherwise, comparisons between the main and hard splits use the same model and Gold-present configuration ($\ell = 3, k = 5$).

Behavior-conditioned metrics. Tool-call rate is the fraction of examples with at least one valid executed call. Tool-Acc and NoTool-Acc condition exact-match accuracy on having or not having such a call; empty conditioning groups are omitted. These model-selected groups describe invocation behavior, not the causal effect of calling tools, and success with a call need not rely on its output.

4.4.1 TOOL CONNECTIVITY AS MAXIMUM DEPENDENCY DEPTH

For trajectory τ, form a directed acyclic graph $G_\tau = (V_\tau, E_\tau)$ whose nodes are valid executed calls and their observations. An edge (v_j, v_i) exists when an earlier output is verified through its trace provenance to supply an input to the later call. Node depth and maximum trajectory depth are

$$d(v_i) = 1 + \max\big(\{d(v_j) : (v_j, v_i) \in E_\tau\} \cup \{0\}\big), \tag{1}$$

$$D(\tau) = \max\big(\{d(v_i) : v_i \in V_\tau\} \cup \{0\}\big). \tag{2}$$

Depth counts nodes on the longest verified path: no calls give $D = 0$, independent calls $D = 1$, and a chain of r dependent calls $D = r$. For each model and fixed split/catalog condition, we report answer accuracy among trajectories with $D(\tau) = d$ at each observed depth $d \geq 1$; no-call outcomes are covered above. Unlike $h(p)$, $D(\tau)$ describes model-chosen execution, so different models' depth bins can contain different questions.

4.5 HUMAN ERROR ANALYSIS

For trace diagnostics, we sample failed traces and assign non-exclusive failure labels using full reasoning/tool-call trajectories. The annotation protocol, taxonomy, and marginal failure distributions are reported in Appendix J, while co-occurrence patterns are analyzed in Section 5.6.

5 RESULTS AND ANALYSIS

We vary distractor–gold similarity (Levels 1–5), budget k, and logical-hop complexity under the main Plan+ReAct protocol (Section 4.3).

Gold-present difficulty trends. Appendix F provides representative accuracy curves by logical-hop count. Accuracy decreases with logical-hop count even when gold tools are available, indicating that tool availability alone does not ensure successful long-horizon execution. High-similarity distractors further amplify this degradation, especially in higher-hop regimes. Thus, logical-hop complexity and distractor similarity expose complementary sources of difficulty under controlled tool availability.

Table 1: Main-split behavior under Gold-only, Gold-present, and Distractors-only catalogs. Both distractor conditions use Level 3 sampling and $k = 5$; metrics follow Section 4.4. All values are percentages.

Model	Condition	Accuracy	Tool-Acc	NoTool-Acc	Tool-call rate
Claude Sonnet 4.6	Gold-only	98.4	97.9	99.5	65.6
Claude Sonnet 4.6	Gold-present	97.4	96.5	99.4	67.8
Claude Sonnet 4.6	Distractors-only	97.8	98.1	97.8	15.2
Gemini 3.1 Pro	Gold-only	99.2	90.4	99.7	5.4
Gemini 3.1 Pro	Gold-present	99.0	97.7	99.3	18.9
Gemini 3.1 Pro	Distractors-only	98.9	99.4	98.8	17.6
GPT-5	Gold-only	99.2	100.0	99.2	3.9
GPT-5	Gold-present	99.2	100.0	99.2	5.1
GPT-5	Distractors-only	99.0	100.0	99.0	0.3
GPT-4o-mini	Gold-only	54.7	58.8	28.3	86.4
GPT-4o-mini	Gold-present	51.6	55.7	25.9	86.2
GPT-4o-mini	Distractors-only	47.2	50.8	39.7	67.2
Llama 3-8B	Gold-only	21.6	23.7	0.2	90.7
Llama 3-8B	Gold-present	13.8	15.8	1.4	86.4
Llama 3-8B	Distractors-only	8.1	8.7	0.2	92.9
Qwen 2.5-7B	Gold-only	57.6	60.9	31.0	89.1
Qwen 2.5-7B	Gold-present	58.4	60.5	52.0	75.6
Qwen 2.5-7B	Distractors-only	51.8	52.1	51.2	69.0

5.1 Behavior-conditioned tool-use profiles

Table 1 reports main-split accuracy, Tool-Acc, NoTool-Acc, and tool-call rate using the definitions in Section 4.4. These metrics distinguish invocation behavior without establishing that a called tool caused success.

ToolMATH distinguishes tool-use policies behind similar final accuracy. The frontier models maintain high accuracy across catalogs, but GPT-5 rarely calls tools, especially in Distractors-only, and its observed Tool-Acc stays high. Claude Sonnet 4.6 calls frequently in Gold-only and Gold-present while maintaining high Tool-Acc. Gemini 3.1 Pro lies between these regimes, increasing invocation with distractors while preserving strong accuracy.

GPT-4o-mini, Llama 3-8B, and Qwen 2.5-7B invoke tools more frequently but their Tool-Acc drops as catalogs become less reliable, especially in Distractors-only. Llama 3-8B calls in over 90% of Distractors-only instances with Tool-Acc below 10%, exposing poor calibration to available capabilities.

5.2 Distractors, Strict Equivalence, and Functional Substitution

Our operational audit detects 797 strictly equivalent pairs involving 1,027 tools (8.30% of the pool); alternatives detected as equivalent to a problem's gold tools are excluded from its distractor catalog. The complete audit and cross-problem invocation statistics are in Appendix L.2. A non-equivalent tool may implement one intermediate step, operate on a different representation, or participate in a different solution method. Appendix H illustrates this by composing vector additions for complex arithmetic. Retaining this partial substitutability reflects realistic tool overlap: success without gold tools can involve alternative tool composition or internal reasoning, which the invocation-conditioned analysis below distinguishes.

5.3 Adaptability and Robustness under tool-catalog shifts

To measure success after catalog changes, let S_m be the questions model m answers correctly under Gold-only, and $Y^C_{m,\ell}(q)$ indicate correctness under target condition C and level ℓ. The performance retention ratio (PRR) is

$$\mathrm{PRR}^C_{m,\ell} = \frac{1}{|S_m|} \sum_{q \in S_m} Y^C_{m,\ell}(q).$$

Table 2: Adaptability decomposed by invocation behavior on the main split. Each model is evaluated on its own Gold-only-success subset under Level 1 Distractors-only catalogs with $k = 5$. All values are percentages; the call rate r_m uses this same subset.

Model	Adaptability A_m	With call A_m^+	No call A_m^0	Call rate r_m
GPT-4o-mini	59.68	40.77	68.29	31.29
Llama 3-8B	8.63	3.59	52.11	89.61
Qwen 2.5-7B	65.12	52.12	69.26	24.15

Table 3: Main-split Adaptability (Level 1) and Robustness across Levels 1–5 (percentages), with $k = 5$ throughout and mean $\pm$ standard deviation. Table 2 decomposes Adaptability by invocation.

Model	**Adaptability**	**Rob.(L1)**	**Rob.(L2)**	**Rob.(L3)**	**Rob.(L4)**	**Rob.(L5)**	**Robustness Avg.**
GPT-4o-mini	59.68	89.95	89.64	89.44	90.84	89.61	89.90 ± 0.56
Llama 3-8B	8.63	66.07	67.65	66.82	77.20	72.95	70.14 ± 4.78
Qwen 2.5-7B	65.12	94.29	94.81	90.32	89.55	86.91	91.18 ± 3.33

Adaptability ($\mathrm{PRR}^{\mathrm{DO}}_{m,1}$) measures retention with only Level 1 distractors; *Robustness* ($\mathrm{PRR}^{\mathrm{GP}}_{m,\ell}$) measures retention with distractors added alongside gold tools. We report Robustness level-wise and its mean $\pm$ standard deviation across Levels 1–5. Level 1 may still contain useful non-equivalent tools, so Adaptability alone does not distinguish internal reasoning from alternative tool use.

Conditioning Adaptability on tool invocation. Partition S_m into S_m^+ (at least one valid call) and S_m^0 (no valid call), using behavior in the Level 1 *Distractors-only* trajectory. Writing $Y_m^{\mathrm{DO}} = Y_{m,1}^{\mathrm{DO}}$, define

$$A_m^+ = \frac{\sum_{q \in S_m^+} Y_m^{\mathrm{DO}}(q)}{|S_m^+|}, \qquad A_m^0 = \frac{\sum_{q \in S_m^0} Y_m^{\mathrm{DO}}(q)}{|S_m^0|}, \tag{3}$$

$$r_m = \frac{|S_m^+|}{|S_m|}, \qquad A_m = r_m A_m^+ + (1 - r_m) A_m^0. \tag{4}$$

The conditional accuracies A_m^+, A_m^0 contribute $r_m A_m^+$ and $(1 - r_m)A_m^0$ to aggregate Adaptability. Empty conditional groups are omitted; retention is undefined if S_m is empty. All denominators in Table 2 use Gold-only successes, unlike the full-split metrics in Table 1. Appendix L.3 retains the set-based formulation and full conventions.

All three models are more accurate on their no-call subset. GPT-4o-mini and Qwen 2.5-7B invoke tools on only 31.29% and 24.15% of these Gold-only-success questions, so much of their retained success is no-tool fallback rather than demonstrated distractor substitution. Llama 3-8B invokes tools in 89.61% of cases but reaches only 3.59% conditional accuracy on those trajectories; its 52.11% no-call accuracy applies to the remaining 10.39%. The decomposition therefore exposes whether failures occur predominantly with attempted tool use or without it.

An incorrect no-call answer is compatible with insufficient internal reasoning but does not prove that avoidance was optimal. With-call errors may involve selection, arguments, execution, observation use, or subsequent reasoning, rather than necessarily a confusing distractor. The trace taxonomy in Appendix J examines these alternatives.

Table 3 shows that Qwen 2.5-7B has the highest Adaptability (65.12%), largely from its no-call subset, and average Robustness (91.18 ± 3.33), with a decline toward more similar distractors. GPT-4o-mini retains slightly less success (Adaptability 59.68%) but is especially stable across levels (89.90 ± 0.56). Llama 3-8B has much weaker Adaptability (8.63%) and Robustness (70.14 ± 4.78); its higher Robustness at Levels 4–5 suggests less disruption by highly similar alternatives despite poor overall retention. These profiles distinguish how success is preserved or disrupted by changes in tool availability and catalog difficulty.

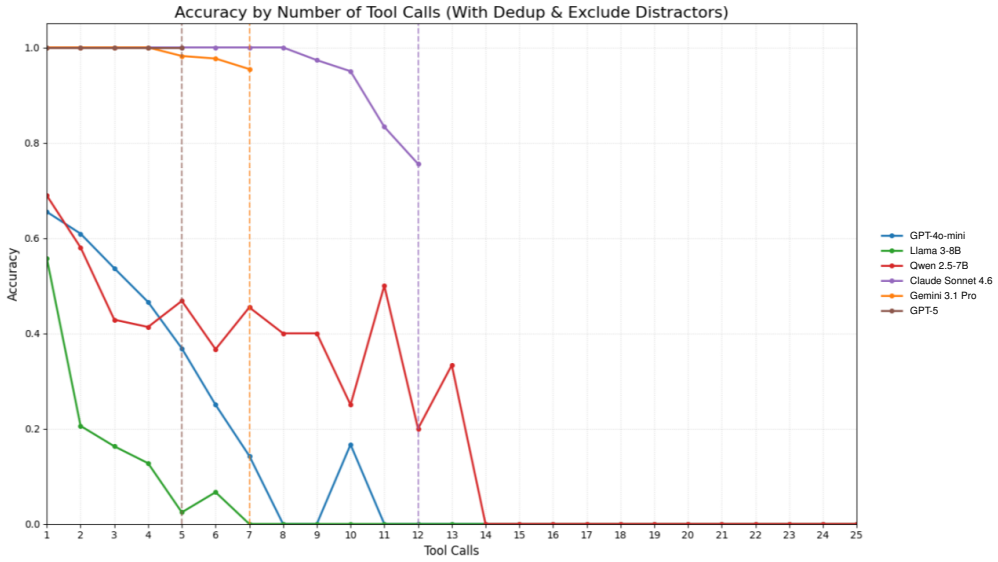


Figure 2: **Answer accuracy by maximum tool-dependency depth.** Main-split Plan+ReAct trajectories under Gold-present catalogs with $(\ell = 3, k = 5)$. The horizontal axis gives the length of the longest verified output-to-input dependency path, and the vertical axis gives accuracy among trajectories at that depth. Vertical dashed lines mark the endpoints of the displayed frontier-model curves.

5.4 Tool Connectivity Through Verified Dependencies

We evaluate Tool Connectivity through answer accuracy conditioned on the maximum dependency depth $D(\tau)$ defined in Section 4.4.1. Figure 2 compares the six models on the main split under Plan+ReAct with Gold-present catalogs, Level 3 distractors, and $k = 5$.

GPT-4o-mini and Llama 3-8B lose accuracy rapidly as dependency depth increases. Llama 3-8B already falls to approximately 20% accuracy at depth two and remains below GPT-4o-mini across depths two through five. Qwen 2.5-7B follows a less monotonic profile: its intermediate-depth accuracy fluctuates, with local recoveries around depths seven and eleven. These recoveries do not form a sustained high-accuracy regime.

Frontier models remain substantially more accurate over the depths covered by their displayed curves. GPT-5 stays at 100% through depth five, and Gemini 3.1 Pro remains above 95% through depth seven. Claude Sonnet 4.6 maintains 100% through depth eight before declining to approximately 76% at depth twelve. Claude's decline shows that accuracy can deteriorate as dependencies deepen even for a model that is highly reliable at shorter depths.

Endpoint markers delimit observed ranges: GPT-5's shorter curve does not establish performance at the depths covered by Claude Sonnet 4.6. Its observed curve stays at ceiling-level performance, but this does not necessarily mean that GPT-5 is the best model at executing long connected tool chains. Rather, GPT-5 appears to adopt a highly conservative tool-use policy: when a problem would otherwise require a longer sequence of tool calls, it often solves the problem directly or minimizes tool use instead of extending the tool-call trajectory. Thus, the absence of later observations should be read as part of the model's behavior—a tendency to avoid long tool-use chains—rather than as evidence that it would maintain the same accuracy if forced to execute them.

5.5 ToolMATH-Hard: tool availability and difficulty

ToolMATH-Hard complements automatic construction with human-authored tools and longer reference-solution chains (Section 3.2). For GPT-4o-mini, Llama 3-8B, and Qwen 2.5-7B, the Gold-only condition separates more clearly from No tools and Distractors-only on this split, especially at higher annotated hops (Appendix I). Table 4 additionally compares frontier models under

Table 4: Frontier models under Gold-present ($\ell = 3, k = 5$). Main and HARD accuracy and call rates use their full splits (HARD: 329 questions); HARD Tool-Acc and NoTool-Acc condition on each model's invocation. Values are percentages.

Model	Main Acc.	HARD Acc.	Main Call rate	HARD Call rate	HARD Tool-Acc	HARD NoTool-Acc
Claude Sonnet 4.6	97.8	86.63	67.8	96.96	86.21	100.00
Gemini 3.1 Pro	99.0	93.62	18.9	35.87	93.22	93.84
GPT-5	99.2	91.19	5.1	53.19	89.71	92.86

matched Gold-present settings. All three models have lower accuracy and higher tool-call rates on TOOLMATH-HARD, extending the evaluation beyond the near-ceiling task accuracy observed on the main split. Together with the hop distribution in Table 7, these results support the validity of the hard split as an additional difficulty setting within TOOLMATH.

The split comparison reflects differences in both problem selection and construction, so it does not isolate logical depth as the sole cause of lower accuracy. Same-question controls in Appendix I.2 further show that providing tools does not consistently improve frontier-model accuracy over No tools. We therefore interpret the increased invocation alongside answer accuracy as a diagnostic of tool-use behavior, and not every question is required to call tools to answer correctly.

5.6 TRACE-LEVEL FAILURE DIAGNOSTICS

Aggregate accuracy and Tool-Acc do not reveal how failures are organized inside a trace. We annotate failed trajectories with a non-exclusive taxonomy spanning planning, selection, hallucination, parameterization, formatting, reasoning, observation use, repetition, and incomplete execution (Appendix J). Tool selection errors become more broadly coupled with other errors when distractors are added or gold tools removed, exposing catalog-dependent failure organization beyond final accuracy.

5.7 FRAMEWORK SENSITIVITY

TOOLMATH-HARD also reveals sensitivity to the tool-use controller. Under the Gold-only condition, we compare No tools, ReAct, DFSDT, and Plan+ReAct; detailed curves and heatmaps are provided in Appendix K. The main trend is that controller differences are modest at low hops but become clearer at higher hops, where explicit planning more consistently preserves accuracy. This indicates that TOOLMATH can diagnose not only model-level tool-use behavior, but also the effect of controller design on long-horizon tool execution.

6 CONCLUSION

We introduced TOOLMATH, a math-grounded diagnostic benchmark for long-horizon tool use under systematic tool-catalog constraints. Its construction methodology combines reusable tool extraction, validation, and human repair/authoring to model connected execution, overlapping functionality, and missing capabilities. Gold-only, Gold-present, and Distractors-only conditions support Adaptability, Robustness, and verified dependency-depth diagnostics. Together with behavior-conditioned metrics and failure co-occurrence analysis, TOOLMATH shows that models with similar final accuracy can differ substantially in whether they avoid tools, use them reliably, succeed through adaptive substitution, over-call under missing capabilities, or fail through coupled planning and tool-selection errors.

The pipeline incorporates safeguards for its construction choices: multi-model validation and human repair reduce specification noise, strict-equivalence screening excludes detected duplicates, and TOOLMATH-HARD broadens coverage beyond successful automatic trajectories. The human-authored route is thus part of the methodology for constructing usable evaluation environments when automatic construction and validation are insufficient. The benchmark models these catalog properties within mathematics; transfer to general APIs, residual selection effects, and the non-causal interpretation of model-chosen trajectories remain scope considerations (Appendix M).

AI USE STATEMENT

In this work, we used generative AI tools for code generation and assistance. All AI-generated code used in the project was reviewed, verified, and tested for correctness by the authors. We also used generative AI tools for writing suggestions and error correction in the manuscript; all AI-assisted revisions were reviewed by the authors. Additionally, we used large language models as part of the research methodology for generating tools in the TOOLMATH main split, validating generated tools and benchmark instances, and using LLMs as evaluated systems in the benchmark experiments. In particular, LLM-generated tools were subjected to automated validation and, where necessary, human audit and repair; model-based validation results were aggregated according to the procedures described in the paper. The final benchmark construction, experimental design, analysis, and interpretation of results were reviewed by the authors. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, code, benchmark artifacts, or other materials produced with the aid of generative AI.

ETHICS STATEMENT

TOOLMATH is a benchmark for evaluating tool-augmented language models under controlled multi-tool conditions. It aims to improve the reliability and interpretability of tool-use evaluation by using correctness-checkable tasks, validating tool specifications, and stress-testing models under tool-list redundancy and long-horizon execution. This can help surface brittle behaviors and support safer deployment of tool-augmented systems. Human annotation supports benchmark curation, tool construction, and trace-level error analysis of mathematical problems and model-generated executions. The resulting scores should be interpreted within this evaluation scope and do not by themselves establish the reliability of a deployed tool-using system. Source attribution and dataset licensing are documented in Appendix A.

REPRODUCIBILITY STATEMENT

We document tool extraction and logical-hop annotation in Appendix B, validation and human repair in Appendix C, and deterministic distractor construction in Appendix D. Appendix E provides the evaluation prompts, execution rules, budgets, and decoding settings; Section 4.4.1 defines the dependency-depth metric. Detailed results across catalog settings and execution frameworks are provided in Appendices F–K. Code for benchmark construction and evaluation, together with the benchmark artifacts, is maintained in the project repository at `https://anonymous.4open.science/r/ToolMATH-7B71`.

REFERENCES

Marah Abdin, Samir Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, et al. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. *CoRR*, abs/2404.14219, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems. *CoRR*, abs/2110.14168, 2021.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR, 2023. arXiv:2211.10435.

Aaron Grattafiori et al. The Llama 3 Herd of Models. *CoRR*, abs/2407.21783, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1*, 2021. arXiv:2103.03874.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B. *CoRR*, abs/2310.06825, 2023.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tenenholtz. MRKL Systems: A Modular, Neuro-Symbolic Architecture that Combines Large Language Models, External Knowledge Sources and Discrete Reasoning. *CoRR*, abs/2205.00445, 2022.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, 2023. doi:10.18653/v1/2023.emnlp-main.187; arXiv:2304.08244.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large Language Model Connected with Massive APIs. In *Advances in Neural Information Processing Systems*, 2024. Poster; arXiv:2305.15334.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *International Conference on Machine Learning (ICML) Poster Track*, 2025. OpenReview.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. TOOLLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024. Spotlight; arXiv:2307.16789.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems 36*, 2023. arXiv:2302.04761.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. In *Advances in Neural Information Processing Systems 36*, 2023. arXiv:2303.17580.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. TaskBench: Benchmarking Large Language Models for Task Automation. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2311.18760.

Eduardo Treviño, Hugo Contant, James Ngai, Graham Neubig, and Zora Zhiruo Wang. Benchmarking Failures in Tool-Augmented Language Models. In *Annual Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*, 2025. arXiv:2503.14227.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the Tool Manipulation Capability of Open-source Large Language Models. *CoRR*, abs/2305.16504, 2023.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 Technical Report. *CoRR*, abs/2412.15115, 2024.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 Technical Report. *CoRR*, abs/2407.10671, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. REACT: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.03629.

Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Guoyin Wang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, Kaidong Yu, Peng Liu, Qiang Liu, Shawn Yue, Senbin Yang, Shiming Yang, Wen Xie, Wenhao Huang, Xiaohui Hu, Xiaoyi Ren, Xinyao Niu, Pengcheng Nie, Yanpeng Li, Yuchi Xu, Yudong Liu, Yue Wang, Yuxuan Cai, Zhenyu Gu, Yudong Liu, Zhiyuan Liu, Zonghong Dai, and 01.AI. Yi: Open Foundation Models by 01.AI. *CoRR*, abs/2403.04652, 2024.

A SOURCE DATASET AND LICENSE

TOOLMATH is derived from the Mathematics Aptitude Test of Heuristics (MATH) dataset (Hendrycks et al., 2021). We use the Hugging Face distribution `qwedsacf/competition_math`, which contains competition mathematics problems and step-by-step solutions from sources such as AMC 10, AMC 12, and AIME. Each example includes a problem statement, a worked solution, a difficulty level, and a problem type. The Hugging Face dataset card links its licensing information to the original Hendrycks MATH repository, whose license is the MIT License. We cite the original MATH paper and use the dataset only as the source corpus for constructing derived tool-grounded benchmark instances.

B TOOL EXTRACTION AND LOGICAL-HOP ANNOTATION DETAILS

This appendix describes (i) how we extract reusable tools from annotated MATH solutions, (ii) the tool schema and storage format, and (iii) how we annotate each problem with a logical-hop count used for evaluation.

B.1 TOOL EXTRACTION PROMPT

We extract tools by prompting a function generator model to return a list of reusable helper functions as a JSON array. The prompt encourages multiple small functions (rather than monolithic solutions) and constrains parameter types to a fixed whitelist to simplify downstream tool schemas.

Prompt template (verbatim).

```
You are a Python function-generator for a math-solver system.

**Task**
Extract every *reusable* helper function that appears implicitly in
    the
following solved problem.  Each function should capture a general
mathematical operation.  Aim for **multiple small functions** if the
solution has several distinct steps.

**Output format**
Return a **JSON array**.  Each element is an object with **exactly**
    these keys:
- "name"        - concise snake_case identifier
- "description" - what the function does and parameter informations.
- "inputs"      - dict {"param": "<python_type>"}
                  allowed types: <WHITELISTED_TYPES>
- "function"    - full Python `def` (no explanations)

Example (array with two functions):
[
  {
    "name": "common_divisors",
    "description": "Return sorted list of common divisors of two
    integers a and b.",
    "inputs": {"a": "int", "b": "int"},
    "function": "def common_divisors(a: int, b: int):\\n    ...return
    divs"
  },
  {
    "name": "integer_points_in_interval",
    "description": "Number of integers in the open interval (lo, hi],
    where lo and hi are integers.",
    "inputs": {"lo": "int", "hi": "int"},
    "function": "def integer_points_in_interval(lo: int, hi: int):\\n
       ...return count"
```

```
  }
]

If **no reusable function** can be identified, output an *empty array*
     `[]`.
Do NOT wrap the JSON in markdown fences.
---
Problem:
{problem}

Solution:
{solution}
```

B.2 Tool Schema and Storage Format

Each extracted tool is represented by a tuple consisting of a tool identifier, natural-language documentation, a typed input schema, and a Python implementation. For evaluation, models only observe the *name*, *description*, and *input schema*; the Python code is hidden and executed by the environment.

In our stored JSON, each tool record includes provenance fields that link back to the originating MATH problem and its official solution. In the stored artifact, the `function` field points to a separate `.py` file containing the implementation.

Schema example (truncated).

```
[
  {
    "name": "absolute_difference",
    "description": "Calculate the absolute difference between two
    integers. Parameters: x (int), y (int). Returns: int.",
    "inputs": {
      "x": "int",
      "y": "int"
    },
    "function": "ao5wa0.py",
    "source_problem": "Two arithmetic sequences $A$ and $B$ both begin
     with 30 ... What is the absolute value of the difference between
    the 51st term of $A$ and the 51st term of $B$?",
    "source_solution": "The $n$th term is $a_n=a_1+d(n-1)$. ...
    Substituting $n=51$ ... is $\\boxed{1000}$."
  },
  ...
]
```

B.3 Logical-Hop Annotation

We annotate each problem with a *logical-hop* count summarizing sequential reasoning groups in its reference solution. This problem-level annotation differs from the maximum dependency depth of a model-generated tool trace. Hop groups account for solution steps that can be carried out in parallel.

Step extraction. For each (problem, official solution), we first prompt a model to extract a linearized list of high-level solution steps. These steps are written in natural language and are not tied to any single tool implementation.

Parallelization check and hop construction. We then build hop groups iteratively. Starting from the first step, we maintain a list of existing hop groups. For each new step, we prompt a model to judge whether the step can be parallelized with any existing hop group (i.e., it does not require outputs from that group). If it can be parallelized, we assign it to that group and do not increase the hop count.

Otherwise, we create a new hop group and increase the hop count by one. In evaluation plots, we report hop groups hop 1–7 and bucket all hops ≥ 8 as `hop8+`.

Prompt for step extraction.

```
You are a careful math-solution analyst.
Given the problem and the official solution, extract an ordered list
    of
high-level solution steps. Output JSON only:
{"steps": ["...", "...", ...]}
```

Prompt for parallelization check.

```
You are analyzing whether a new solution step depends on prior steps.
Given:
(1) A list of existing hop groups (each group is a set of steps that
    can run in parallel)
(2) A candidate new step
Decide whether the candidate can be placed into any existing hop group
without depending on outputs from that group.
Output JSON only:
{"parallelizable": true/false, "group_index": <int or -1>}
```

C Multi-model Tool-wise Validation Details

This appendix describes the multi-model tool-wise validation procedure used to check whether a tool's natural-language description and input schema match its executable behavior. The validation task follows the original judgment-based protocol: validator models do not predict replacement outputs, but instead decide whether the implementation's actual output is correct for each test case.

Rule-based test input generation. For each tool, we generate five schema-valid inputs using rule-based templates according to the declared parameter types. For numeric parameters, we include small integers, boundary-like values, and randomly sampled values within safe execution ranges. For list-valued parameters, we sample short lists with valid element types and include edge cases such as singleton or repeated-value lists when applicable. The tool implementation is then executed on each input to obtain the actual output.

Judgment-based validation. Each validator model receives the tool description, input schema, concrete inputs, and actual outputs returned by the implementation. For each test case, the validator judges whether the actual output is correct according to the described semantics of the tool. We use three validator models: GPT-4o, Llama 3-8B, and Qwen 2.5-7B.

Majority validation rule. For each test input x_j, let $y_j = f_t(x_j)$ denote the output produced by the tool implementation. Each validator model m produces a binary judgment

$$z_j^{(m)} \in \{\text{correct}, \text{incorrect}\},$$

where the judgment indicates whether y_j is consistent with the tool description and input schema. We aggregate the three judgments by majority vote:

$$z_j = \text{Majority}\left(z_j^{(\text{GPT-4o})}, z_j^{(\text{Llama3})}, z_j^{(\text{Qwen2.5})}\right).$$

A test case passes if

$$z_j = \text{correct}.$$

A tool is retained only if all five test cases pass. We allow small floating-point tolerance and benign formatting differences, such as `int` versus `float` when numerically equal, or list versus tuple for the same sequence.

Validator prompt.

```
You are validating the behavior of a small deterministic math utility
    function.
You are given its natural-language description and several testcases.
Each testcase has:
- concrete inputs (already converted to the right Python types), and
- the actual_output returned by the implementation.

For EACH case, you must decide whether the actual_output is correct,
based on what the function is *supposed* to do from the description.

Guidelines:
- Carefully infer the intended behavior from the description (and
    input schema if needed).
- Compute or reason about the correct output for each testcase.
- Compare that internal expected result to actual_output.
- Allow small floating-point rounding errors (~1e-6) and harmless
    formatting differences
  (e.g., extra parentheses, int vs float when numerically equal, tuple
     vs list for the same sequence).
- If the implementation's result is clearly wrong (wrong sign, wrong
    magnitude, wrong structure),
  mark the case as incorrect.
```

C.1 Validation and Final Dataset Composition

TOOLMATH applies a sequential validation-and-repair pipeline to ensure that benchmark errors are not conflated with model errors.

Step 1: Tool-wise validation. We validate each extracted tool in isolation to ensure that its natural-language description and input schema are consistent with its executable behavior. For every tool, we generate five schema-valid test inputs using a rule-based input generator that covers nominal, boundary, and randomly sampled values for the declared input types, and execute the tool implementation to obtain actual outputs. We then provide the tool metadata (description and input schema), the concrete inputs, and the actual outputs to three validator models, **GPT-4o**, **Llama 3-8B**, and **Qwen 2.5-7B**. Each validator judges whether the actual output is correct according to the described tool semantics. For each test case, we aggregate the three binary judgments by majority vote. A test case passes if the majority judgment is correct, and a tool passes tool-wise validation only if all five test cases pass. Tools whose outputs are judged incorrect by the majority on at least one test case are discarded from the tool-wise validated set and handled by the downstream repair/audit stage. This procedure reduces reliance on a single LLM judge while preserving the original validation task: validators do not generate replacement answers, but instead judge whether the implementation output matches the natural-language specification. The validation prompt and normalization details are provided in Appendix C.

Step 2: Question-wise validation (trace-based). After tool-wise validation, we perform question-wise validation to identify tools that are *empirically usable* under our tool-use protocol, and to filter questions that admit at least one successful tool invocation in a correct solution. For each MATH problem p, we provide each validator model with (i) the problem statement and (ii) its gold tool set $\mathcal{G}(p)$ (tool name, description, and input schema; code hidden), and run the same Plan+ReAct prompt that will be further used in the main evaluation. A tool $t \in \mathcal{G}(p)$ is marked as *question-wise validated* if there exists at least one validator model that (a) produces a correct final answer for p and (b) successfully calls t at least once in its trace (i.e., selecting t from the provided list, supplying schema-valid arguments, and executing without runtime failure). A question p is marked as *validated* if at least one tool in $\mathcal{G}(p)$ becomes validated by this criterion. This trace-based criterion reflects our intended setting where models may combine internal reasoning with tool execution, rather than relying on tools for every step.

Validator set. We use a fixed validator set: $\mathcal{M} = \{$GPT-4o-mini, Llama 3-8B, Mistral-7B, Qwen2-7B, Qwen 2.5-7B, Phi-3 Medium, and Yi 1.5-9B$\}$.

Concretely, let $\mathcal{C}(p, m)$ be the set of tools successfully executed by model m while solving p correctly. Then the question-wise validated tool set is

$$\mathcal{T}_{\mathrm{qw}} = \bigcup_{p} \bigcup_{\substack{m \in \mathcal{M} \\ \mathrm{Correct}(p,m)=1}} \mathcal{C}(p, m),$$

and a question is retained if $\mathcal{G}(p) \cap \mathcal{T}_{\mathrm{qw}} \neq \emptyset$.

Step 3: Targeted human audit and tool repair. For episodes rejected by question-wise validation, we perform targeted human inspection. If an episode is solvable, we reinstate it. Otherwise, we revise the responsible tool(s) by editing descriptions and/or implementations, then re-run tool-wise validation for the modified tools and re-run question-wise validation for affected episodes. Episodes that remain invalid after this repair loop are excluded from the automatically constructed main split. The human-curated TOOLMATH-HARD recovers cases for which automatic construction and validation were unsuccessful, using separately authored gold tools as described in Section 3.2.

D DISTRACTOR SAMPLING PROCEDURE AND SAFEGUARDS

This appendix details distractor construction after gold-instance and strict-equivalence exclusions, nested inclusion across budgets, and safeguards against tool-identity collisions.

D.1 GLOBAL POOL AND PER-PROBLEM EXCLUSIONS

Global pool. All distractors are sampled from the same global pool $\mathcal{U}$, defined as the union of *filtered and validated* tools (i.e., tools that survive the tool-wise and question-wise filtering/repair pipeline). This ensures distractors are executable and description-consistent under our validation criteria.

Gold removal and strict-equivalence exclusion. For each problem p, $\mathcal{G}(p)$ denotes its intended gold tools, including separately human-authored tools for TOOLMATH-HARD. We remove gold instances by their deterministic identifiers and also remove candidates detected as strictly equivalent to any of them:

$$\mathcal{U}_{\setminus p} = \mathcal{U} \setminus \big(\mathcal{G}(p) \cup \widehat{\mathcal{E}}(p)\big).$$

Here $\widehat{\mathcal{E}}(p)$ is defined by the operational screening procedure below. All five levels, their retrieval rankings, and their fallback pools use this already filtered pool. The equivalence exclusion is performed before forming the ordered lists, so changing the budget does not reintroduce an excluded alternative.

Operational strict-equivalence screen. Candidate pairs must match both the multiset of declared input types and the output type. Within each type-matched group, we embed tool descriptions using `text-embedding-3-large`, retrieve pairs with cosine similarity at least 0.95, and ask GPT-5 whether their descriptions specify the same semantic operation. The type criterion permits a correspondence between argument positions; different names or a different parameter order are not enough to establish different functionality. Pairs judged equivalent under this procedure are excluded when one member is a gold tool for the current problem. Table 8 reports the audit of the 12,369-tool main pool. Similarity filtering and semantic judgments may miss equivalent implementations, so this procedure is a reproducible operational screen, not a proof of exhaustive semantic deduplication. Partially substitutable operations that do not satisfy the equivalence criterion remain eligible.

Problem category. We assign each problem a broad category $c(p)$ (e.g., Algebra, Geometry) using the dataset-provided `type` field. Each tool inherits a category from its `source_problem`. We use $c(\cdot)$ only to define candidate pools for Levels 1 and 3.

D.2 LEVEL 1–3: CATEGORY-CONTROLLED RANDOM SAMPLING

For each problem p, we precompute an *ordered* distractor list of length 100 for each level. Randomness is reproducible via a fixed global seed and a fixed serialized order of $\mathcal{U}$.

Candidate pools. We define:

$$\mathcal{U}_{\setminus p}^{\text{same}} \;=\; \{t \in \mathcal{U}_{\setminus p} : c(t) = c(p)\}, \qquad \mathcal{U}_{\setminus p}^{\text{diff}} \;=\; \{t \in \mathcal{U}_{\setminus p} : c(t) \neq c(p)\}.$$

Level 1 (exclude same-category). We sample 100 tools uniformly from $\mathcal{U}_{\setminus p}^{\text{diff}}$. If $\mathcal{U}_{\setminus p}^{\text{diff}}$ is too small, we fall back to sampling from $\mathcal{U}_{\setminus p}$ to ensure a full-length list.

Level 2 (pure random). We sample 100 tools uniformly from $\mathcal{U}_{\setminus p}$.

Level 3 (same-category). We sample 100 tools uniformly from $\mathcal{U}_{\setminus p}^{\text{same}}$. If $\mathcal{U}_{\setminus p}^{\text{same}}$ is too small, we fall back to $\mathcal{U}_{\setminus p}$.

With/without replacement. If the chosen pool has at least 100 candidates, we sample *without* replacement. Otherwise, we sample *with* replacement (cycling as needed) to produce exactly 100 distractors per level.

D.3 LEVEL 4: EMBEDDING SIMILARITY RETRIEVAL (TOP-100)

Level 4 selects distractors that are semantically close to the gold tools.

Embedding representation. Each tool t is embedded using the OpenAI embedding model `text-embedding-3-large`. The embedded text is the concatenation of tool name and description:

$$x(t) \;=\; \texttt{name}(t) \,\|\, \texttt{description}(t).$$

Embeddings are ℓ_2-normalized, and we use a cosine similarity.

Gold-to-candidate similarity score. For a candidate $t \in \mathcal{U}_{\setminus p}$, we define its similarity to the gold set as

$$s_4(t; p) \;=\; \max_{g \in \mathcal{G}(p)} \cos\big(e(t), e(g)\big),$$

where $e(\cdot)$ is the normalized embedding. We then rank all candidates in $\mathcal{U}_{\setminus p}$ by $s_4(t; p)$ in descending order and take the top 100 as the ordered Level-4 distractor list. If $|\mathcal{U}_{\setminus p}| < 100$, we cycle the ranked list to reach length 100.

Caching and efficiency. We cache tool embeddings and optionally cache top-K nearest neighbors per tool to avoid recomputing full similarity against the entire pool. This does not change the ranking definition above; it only accelerates computation.

D.4 LEVEL 5: KEYWORD OVERLAP + EMBEDDING SIMILARITY (LEXICAL-FIRST RANKING)

Level 5 prioritizes tools that are both lexically and semantically similar to the gold set.

Keyword extraction. We define a fixed keyword vocabulary of common mathematical terms (optionally extended with a small manual list). For each tool t, we extract keywords $K(t)$ by tokenizing `name`(t) and `description`(t) and retaining tokens that appear in the vocabulary. We also form the gold keyword union $K(\mathcal{G}(p)) = \bigcup_{g \in \mathcal{G}(p)} K(g)$.

Lexical overlap score. For each candidate $t \in \mathcal{U}_{\setminus p}$, define

$$o(t; p) \;=\; \big|K(t) \cap K(\mathcal{G}(p))\big|.$$

Two-stage ranking. We rank candidates by the tuple $(o(t;p),\ s_4(t;p))$ in descending order (lexical overlap first, embedding similarity as tie-breaker). Candidates with zero overlap remain eligible but are ranked below candidates with positive overlap. We then take the top 100 (cycling if needed) as the ordered Level-5 distractor list.

D.5 Nested distractor guarantee across budgets k

For each problem p and level $\ell \in \{1, 2, 3, 4, 5\}$, we precompute an ordered list of 100 distractors,

$$\mathbf{L}_\ell(p) = \big[t_1, t_2, \ldots, t_{100}\big].$$

For a distractor budget $k \in \{5, 10, 20, 50\}$, we define

$$\mathcal{D}_{\ell,k}(p) \ = \ \{t_1, \ldots, t_k\}.$$

This construction guarantees *nested inclusion*: if $k_1 < k_2$, then $\mathcal{D}_{\ell,k_1}(p) \subseteq \mathcal{D}_{\ell,k_2}(p)$. For Levels 1–3, the list order is fixed by a deterministic RNG seed; for Levels 4–5, the order is fixed by deterministic ranking.

D.6 Safeguards and collision handling

No accidental gold overlap. We explicitly remove both $\mathcal{G}(p)$ and $\widehat{\mathcal{E}}(p)$ from the candidate pool before any sampling or retrieval. Implementation-wise, each tool instance is tracked by a deterministic identifier derived from tool metadata (including name, description, and implementation reference), and gold-instance identifiers are excluded from the distractor pool for the same problem.

Name collision safeguard (suffixing). Some extracted tools can share the same base name. To avoid identity collisions in tool registries and to ensure that retrieval/sampling treats tools as distinct callable actions, we enforce unique tool names by appending a suffix, e.g., `solve_linear_a`, `solve_linear_b`, ... This safeguard is applied before constructing distractor pools and remains consistent throughout evaluation. It prevents name collisions; semantic equivalence is handled separately by the exclusion screen.

Determinism and reproducibility. All randomized sampling uses a fixed global seed and a fixed serialized ordering of the validated tool pool. Together with cached embeddings and deterministic sorting, this makes distractor lists reproducible across runs given the same validated tool pool.

E Tool-Use Protocol Prompts and Execution Rules

This appendix specifies the exact prompts and execution rules used to compare **ReAct**, **Plan+ReAct**, and **DFSDT** on ToolMath, including (i) system/user prompts, (ii) tool-call formatting constraints, (iii) stopping criteria and budgets, (iv) retry/diversity rules, and (v) decoding and guardrails.

E.1 Common tool-call interface and formatting

Tool visibility. In all protocols, the model is provided with a list of available tools, each containing *name*, *description*, and a typed *input schema*. Tool implementations are hidden and executed by the environment.

Action JSON constraints. When a tool is called, the model must emit exactly one `Action:` line containing a JSON object of the form:

```
{ "name": "<tool_name>", "arguments": {...} }
```

The `arguments` object must be valid JSON and must conform to the tool's input schema (required keys present; value types match the declared schema). The environment rejects malformed JSON or schema-invalid arguments.

Observation. After an `Action` line, the model must wait for the environment to return an `Observation:` line containing the tool output. The model must condition subsequent reasoning and actions on this observation.

Duplicate-call guardrail. Across protocols, we disallow repeating the exact same (`name`, `arguments`) pair. If the model has already executed an identical call earlier in the same episode, it must reuse the prior observation rather than calling the tool again. This is enforced both as an instruction and by a trace-level cache used in evaluation.

Answer format. All protocols terminate by outputting a single line:

`ANSWER: <numeric answer>`

We use exact-match answer normalization for scoring (Section 4.4).

E.2 REACT-ONLY PROTOCOL

ReAct-only runs without an explicit planning stage. The model interleaves one-line thoughts with optional tool calls.

System prompt (verbatim).

```
You are an expert competition-math solver.
For every turn output *exactly one* line beginning with "Thought:
    ...".
If you need a tool, follow IMMEDIATELY with ONE line:
Action: { "name": "<tool>", "arguments": { ... } }
Then WAIT for the Observation line before thinking again.
Never call a tool with the same arguments twice - reuse the prior
    Observation.
If you have already called a tool with the same name and identical
    arguments, DO NOT call it again; reuse the cached Observation
    verbatim and move forward.
Finish with:
ANSWER: <numeric answer>
```

E.3 PLAN+REACT PROTOCOL

Plan+ReAct consists of two stages: (i) a planner that produces a short structured plan, and (ii) a ReAct-style solver that follows the plan while interacting with tools.

Planner system prompt (verbatim).

```
You are an expert planner for competition-math problems.
Given the problem and available tools, write a concise, structured
    step-by-step plan.
Use a numbered list (1-7 steps). If a tool is needed, mention its
    exact name.
Do not solve the problem or compute final values.
Do not include any numeric answers or intermediate calculations.
Do not include explicit numbers, computed values, or worked examples;
    refer abstractly to the needed computations.
```

Solver system prompt (verbatim).

```
You are an expert competition-math solver.
For every turn output *exactly one* line beginning with "Thought:
    ...".
If your next action or tool choice deviates from the provided plan,
    include the token <PLAN_CHANGED>
in the Thought line for that turn (only when a deviation happens).
If you need a tool, follow IMMEDIATELY with ONE line:
```

```
Action: { "name": "<tool>", "arguments": { ... } }
Then WAIT for the Observation line before thinking again.
Never call a tool with the same arguments twice - reuse the prior
    Observation.
If you have already called a tool with the same name and identical
    arguments, DO NOT call it again; reuse the cached Observation
    verbatim and move forward.
Finish with:
ANSWER: <numeric answer>
```

E.4 DFSDT PROTOCOL (DECISION-TREE SEARCH)

DFSDT follows the decision-tree search style popularized by ToolLLM-like frameworks, where the model proposes a single action at a time and the controller branches over alternatives. Our prompts are adapted from ToolLLM (Appendix A.8) with minimal changes for math tool use.

System prompt (solver; verbatim).

```
You are MathTool-GPT, capable of using provided functions to solve
    math problems.

Workflow:
1) I will give you a math problem.
2) At each step, analyze the current state and choose the next action
    by calling ONE function.
3) You will receive the function result (observation), then repeat.
4) When you have enough to answer, call Finish with return_type="
    give_answer" and include a complete final_answer.
5) If you cannot proceed effectively, call Finish with return_type="
    give_up_and_restart".

Rules:
- Keep your thoughts concise (max 5 sentences).
- Prefer tool calls for computation, algebra, simplification,
    verification.
- Make one attempt per idea; don't repeat the same call with the same
    arguments.
```

User prompt (problem wrapper; verbatim).

```
Task description:
Solve the following problem using the available functions.

Problem:
{problem_text}
```

Diversity user prompt (different child node; verbatim).

```
This is not the first time you try this state; previous attempts from
    here failed.

Here are previous action candidates you already tried from THIS STATE:
{previous_candidate_actions}

Now you MUST choose a next action that is different from all previous
    candidates above.
```

```
First analyze why they failed, then take a new action.
```

E.5 Stopping criteria, budgets, and retry rules

Termination. For ReAct and Plan+ReAct, an episode terminates when the model outputs a line of the form `ANSWER: <numeric answer>`. For DFSDT, the controller terminates when the model calls `Finish` with `return_type="give_answer"` and provides a complete `final_answer`. If a run ends without producing a valid final answer format, it is counted as **no answer** in trace analysis.

Wall-clock timeouts. We enforce a per-question wall-clock timeout of **120 seconds**. Separately, each tool execution is protected by a per-call timeout of **60 seconds**; if exceeded, the observation is set to an error message and the run continues until termination or the question timeout is reached.

Duplicate-call cache and enforcement. We cache tool results keyed by (`name`, `arguments`), using a canonical serialization of `arguments` (sorted keys and normalized numeric forms). If the model repeats an identical call, the environment returns a cached observation instead of re-executing the tool and issues an explicit reminder not to repeat identical actions. After multiple repeated identical actions, further identical actions are ignored and the model is instructed to proceed to the next reasoning step or output `ANSWER`.

Action parsing robustness. If the emitted `Action` JSON is malformed, the environment returns an explicit error observation and continues the loop. If the model produces repeated non-progressing steps (e.g., repeated malformed actions or repeated duplicates), the environment injects a short reminder to follow the one-`Thought`-per-turn constraint and to reuse cached observations.

LLM call retries (API robustness). All LLM calls (planner, solver, and judge) use a retry wrapper that attempts up to **5** retries on transient API errors (e.g., rate limits or timeouts) with exponential backoff (base **0.8**) and jitter. If retries are exhausted, the episode is terminated and counted as incorrect/no-answer depending on whether a valid final answer was produced.

DFSDT restarts and diversity. In DFSDT, when the model emits `Finish(return_type="give_up_and_restart")`, the controller restarts from the root state under the same global budgets. When revisiting the same search state, the controller provides the diversity prompt listing previously tried action candidates from that state and requires the next proposed action to differ from them.

E.6 Decoding, temperatures, and guardrails

Decoding / temperature. We use near-deterministic decoding for the solver across protocols with **temperature** $= 0.0$ to reduce variance in execution traces. In Plan+ReAct, the planner uses **temperature** $= 0.2$ to encourage concise but non-degenerated plan drafts, while the subsequent solver remains at **temperature** $= 0.0$.

Formatting guardrails. All protocols require strict adherence to the one-line `Thought: ...` format per step. If a tool is needed, the model must emit exactly one additional line `Action: { "name": ..., "arguments": {...} }` and then wait for the environment `Observation`. The environment validates that the `Action` payload is parseable (JSON-first, with a permissive fallback) and returns explicit error observations when parsing fails.

Execution guardrails. We enforce (i) per-tool and per-question timeouts, and (ii) cache-based prevention of exact duplicate tool executions, as described in Appendix E.5.

F GOLD-PRESENT DIFFICULTY TRENDS

We aggregate results from the Gold-present condition across distractor budgets k and plot accuracy as a function of logical-hop count, with separate curves for distractor similarity levels (Levels 1–5) and two baselines: the No tools baseline and the Gold-only condition. Figure 3 shows GPT-4o-mini as a representative example.

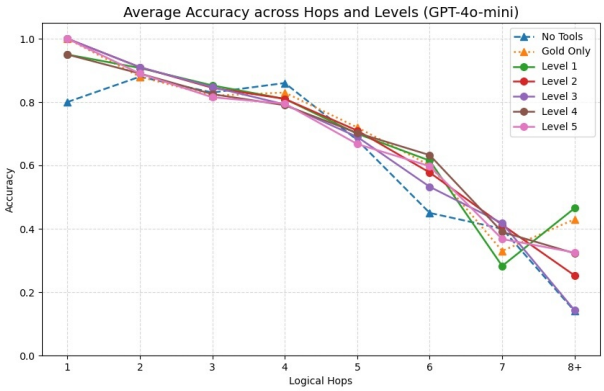


Figure 3: **GPT-4o-mini** accuracy by logical-hop group, averaged over distractor budgets k. Curves show the No tools baseline, the Gold-only condition, and the Gold-present condition with distractor similarity Levels 1–5. Accuracy decreases with hop count; higher-similarity distractors separate most clearly in higher-hop regimes.

Accuracy generally decreases as hop count increases across all similarity levels, and the same trend holds for the No tools baseline. This confirms that hop count captures intrinsic long-horizon, multi-step difficulty beyond effects from tool-set overlap alone. Higher similarity levels are associated with larger performance differences at higher hops: while curves are close at low hops (hop 1–2), high-overlap levels show larger drops at hop 5+ and separate more clearly in the hardest regimes (hop 7 and hop 8+). This interaction suggests that distractor similarity is most informative when evaluated together with long-horizon dependency structure, rather than as an isolated source of difficulty.

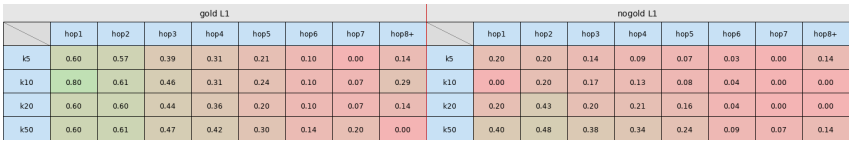
The Gold-only condition remains near the ceiling at low hops and stays above No tools at moderate hop depths, but performance still declines substantially at higher hops. Thus, the availability of gold tools alone does not guarantee success on these reference-solution hop groups; the plot does not establish that tools are necessary for every question or directly measure executed dependency depth. Overall, logical-hop complexity emerges as the dominant difficulty axis, with distractor similarity acting as an amplifying factor rather than the primary driver of failures.

G DETAILED EVALUATION RESULTS FOR THE DISTRACTORS-ONLY CONDITION

Figures 4, 5, and 6 compare Gold-present and Distractors-only accuracy under Level 1 sampling for GPT-4o-mini, Llama 3-8B, and Qwen 2.5-7B, respectively.

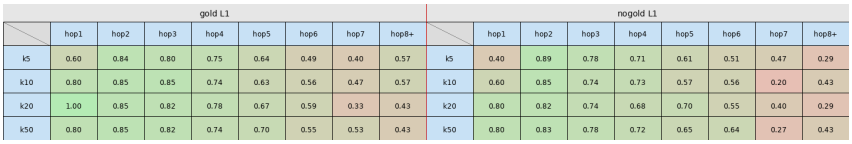
	gold L1									nogold L1								
	hop1	hop2	hop3	hop4	hop5	hop6	hop7	hop8+		hop1	hop2	hop3	hop4	hop5	hop6	hop7	hop8+
k5	0.80	0.92	0.85	0.81	0.74	0.72	0.20	0.43	k5	0.60	0.74	0.67	0.66	0.57	0.46	0.47	0.14
k10	1.00	0.90	0.87	0.79	0.70	0.57	0.33	0.43	k10	0.40	0.73	0.69	0.62	0.59	0.41	0.27	0.29
k20	1.00	0.91	0.83	0.79	0.68	0.60	0.27	0.43	k20	1.00	0.83	0.73	0.67	0.63	0.50	0.40	0.43
k50	1.00	0.90	0.86	0.85	0.68	0.57	0.33	0.57	k50	1.00	0.85	0.77	0.73	0.63	0.54	0.40	0.57

Figure 4: Gold-present condition (Level 1) vs. Distractors-only condition (Level 1) for **GPT-4o-mini** (full grid).



	gold L1									nogold L1								
	hop1	hop2	hop3	hop4	hop5	hop6	hop7	hop8+		hop1	hop2	hop3	hop4	hop5	hop6	hop7	hop8+
k5	0.60	0.57	0.39	0.31	0.21	0.10	0.00	0.14	k5	0.20	0.20	0.14	0.09	0.07	0.03	0.00	0.14
k10	0.80	0.61	0.46	0.31	0.24	0.10	0.07	0.29	k10	0.00	0.20	0.17	0.13	0.08	0.04	0.00	0.00
k20	0.60	0.60	0.44	0.36	0.20	0.10	0.07	0.14	k20	0.20	0.43	0.20	0.21	0.16	0.04	0.00	0.00
k50	0.60	0.61	0.47	0.42	0.30	0.14	0.20	0.00	k50	0.40	0.48	0.38	0.34	0.24	0.09	0.07	0.14

Figure 5: Gold-present condition (Level 1) vs. Distractors-only condition (Level 1) for **Llama 3-8B** (full grid).



	gold L1									nogold L1								
	hop1	hop2	hop3	hop4	hop5	hop6	hop7	hop8+		hop1	hop2	hop3	hop4	hop5	hop6	hop7	hop8+
k5	0.60	0.84	0.80	0.75	0.64	0.49	0.40	0.57	k5	0.40	0.89	0.78	0.71	0.61	0.51	0.47	0.29
k10	0.80	0.85	0.85	0.74	0.63	0.56	0.47	0.57	k10	0.60	0.85	0.74	0.73	0.57	0.56	0.20	0.43
k20	1.00	0.85	0.82	0.78	0.67	0.59	0.33	0.43	k20	0.80	0.82	0.74	0.68	0.70	0.55	0.40	0.29
k50	0.80	0.85	0.82	0.74	0.70	0.55	0.53	0.43	k50	0.80	0.83	0.78	0.72	0.65	0.64	0.27	0.43

Figure 6: Gold-present condition (Level 1) vs. Distractors-only condition (Level 1) for **Qwen 2.5-7B** (full grid).

H REPRESENTATIVE CORRECT TRACES IN THE *Distractors-only* CONDITION

This appendix presents representative episodes that are solved correctly under the *Distractors-only* condition (gold tools removed), where the model executes one or more available distractor tools as part of its solution trace. Our goal is to qualitatively illustrate the kinds of tool executions and compositions that can arise even in tool-insufficient environments.

Selection criterion. We collect episodes that satisfy: (i) evaluation is conducted under the *Distractors-only* tool list, (ii) the normalized final answer is correct, and (iii) the trace contains at least one executed tool call (a parsed `Action` followed by an `Observation`).

Example case for GPT-4o-mini.

```
Problem:
How many vertical asymptotes does the graph of
$y=\dfrac{1}{x^3-3x^2+3x-1}$ have?

Tools (distractor-only;):
- solve_rational_equation_quadratic
    -"description": "Solve equation a x/(x-b) + (c x^2 + d)/x = k.
    Returns real solutions excluding poles."
    -"inputs": {
          "a": "float",
          "b": "float",
          "c": "float",
          "d": "float",
          "k": "float"
        }
- line_coefficients_from_point_direction
    ...
- angle_from_three_points
    ...
- sum_first_digits_of_fraction_sum
    ...
- least_multiple_with_num_divisors
    ...
- max_gcd_minus_a
    ...
- t_for_pairwise_sums_cubic
    ...
- real_part_from_modulus_and_imag
```

```
    ...
- evaluate_quadratic
    ...
- c_from_floor_and_fractional_roots
    ...

Plan:
1. Vertical asymptotes occur at real $x$ values where the denominator
    $x^3-3x^2+3x-1$ equals $0$.
2. To find the real roots of the denominator, use the tool `[TOOL:
    solve_rational_equation_quadratic]`, which numerically returns
    real solutions of a cubic polynomial determined by its parameters.
3. Choose parameters so the tool's internal cubic matches $x^3-3x^2+3x
    -1=0$.
4. Count the number of distinct real roots returned.

Thought: Find real zeros of the denominator via a distractor tool and
    count distinct roots.
Action: {"name":"solve_rational_equation_quadratic","arguments":{"a
    ":-8.0,"b":-1.0,"c":1.0,"d":-1.0,"k":-4.0}}
Observation (solve_rational_equation_quadratic, {"a":-8.0,"b":-1.0,"c
    ":1.0,"d":-1.0,"k":-4.0}): [np.float64(1.0)]
Thought: The distinct real root set is $\{1\}$, so there is exactly
    one vertical asymptote.
ANSWER: 1
```

Example case for Qwen 2.5-7B.

```
Problem:
What is $2a+3b$, where $a=2-i$ and $b=-1+i$?

Tools (distractor-only; names only):
- add_vectors_2d
    - "description": "Add two 2D vectors."
    - "inputs": {
        "v1": "list[float]",
        "v2": "list[float]"
        }
- line_coefficients_from_point_direction
    ...
- monic_quadratic_from_surd_shift
    ...
- count_isosceles_from_lengths
    ...
- solve_c_for_polynomial_value
    ...
- add_polynomials
    ...
- line_from_point_normal_2d
    ...
- simplify_nested_radicals
    ...
- sum_coordinates_intersections_parabolas
    ...
- centroid_distance_from_vertex
    ...

Plan:
1. Rewrite each complex number as a 2D vector of its real and
    imaginary parts:
   $a \mapsto [2,-1]$ and $b \mapsto [-1,1]$.
2. Compute the scaled vectors by repeated addition:
   - $2a = a+a$ using `[TOOL:add_vectors_2d]`.
   - $3b = b+b+b$ using `[TOOL:add_vectors_2d]` twice.
3. Add the results to obtain $2a+3b$ as a 2D vector $[\,\Re,\,\Im\,]$.
4. Convert the final 2D vector back to complex form $\Re+\Im i$.

Thought: Represent $a,b$ as 2D vectors and use repeated vector-
    addition to form $2a+3b$.
Action: {"name":"add_vectors_2d","arguments":{"v1":[2,-1],"v2
    ":[2,-1]}}
Observation (add_vectors_2d, {"v1":[2,-1],"v2":[2,-1]}): [4,-2]
Thought: Now, let's build $3b$ by adding $b$ twice more starting from
    $b+b$.
Action: {"name":"add_vectors_2d","arguments":{"v1":[-1,1],"v2
    ":[-1,1]}}
Observation (add_vectors_2d, {"v1":[-1,1],"v2":[-1,1]}): [-2,2]
Thought: Let's add $2b$ and $b$ to get $3b$.
Action: {"name":"add_vectors_2d","arguments":{"v1":[-2,2],"v2
    ":[-1,1]}}
Observation (add_vectors_2d, {"v1":[-2,2],"v2":[-1,1]}): [-3,3]
Thought: We can now add $2a$ and $3b$.
Action: {"name":"add_vectors_2d","arguments":{"v1":[4,-2],"v2
    ":[-3,3]}}
Observation (add_vectors_2d, {"v1":[4,-2],"v2":[-3,3]}): [1,1]
Thought: We have already obtained the result vector.
ANSWER: 1+1j
```

I Additional Tool-Availability and Split Comparisons

I.1 Split Comparisons for GPT-4o-mini, Llama 3-8B, and Qwen 2.5-7B

This appendix compares ToolMATH and ToolMATH-Hard for GPT-4o-mini, Llama 3-8B, and Qwen 2.5-7B under tool availability and intended-tool removal. Frontier-model comparisons are reported separately in Section 5.5. We compare the No tools baseline, the Gold-only condition, and the Distractors-only condition across logical-hop groups. For all settings involving distractors, we use pure random sampling (Level 2) with $k = 10$ distractors.

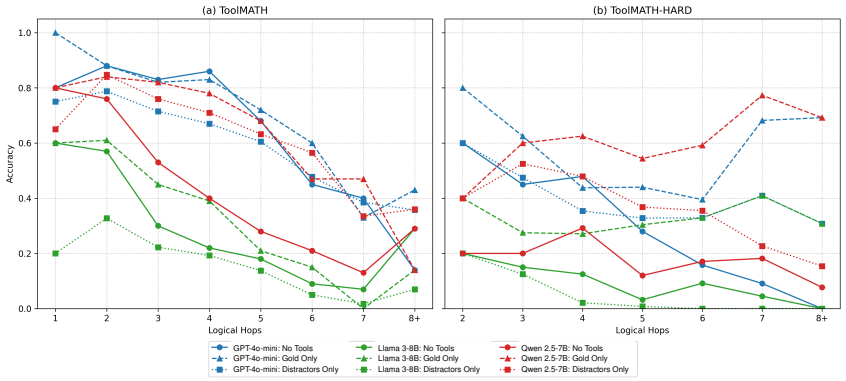


Figure 7: **ToolMATH vs. ToolMATH-Hard: hop-wise accuracy for GPT-4o-mini, Llama 3-8B, and Qwen 2.5-7B.** Left: ToolMATH. Right: ToolMATH-Hard. We report accuracy by logical-hop group under the No tools baseline, the Gold-only condition, and the Distractors-only condition. For all settings involving distractors, we fix the distractor list to pure random sampling (Level 2) with $k = 10$ tools.

Figure 7 shows that the role of intended gold tools differs substantially between the main and hard splits. On the main ToolMATH split, the Gold-only condition usually improves performance over settings without gold tools, but the separation from the Distractors-only condition is often modest. This suggests that many main-split examples remain solvable through internal reasoning or alternative tool-consistent trajectories.

For these three models and this catalog configuration, the hard split shows a larger separation between Gold-only and the settings without intended tools, especially at higher annotated hops. This illustrates sensitivity to intended-tool availability in a more demanding problem set. The result should not be extrapolated to every frontier model: the same-question controls in Table 5 do not show a consistent frontier-model accuracy gain from providing tools. Nor do these reference-hop curves establish the maximum dependency depth of the executed traces.

I.2 Same-Question Frontier-Model Controls on ToolMATH-Hard

The high NoTool-Acc values in Table 4 describe self-selected subsets, so they are not a paired comparison of tool-assisted and unaided solutions. Table 5 instead evaluates No tools, Gold-only, and Gold-present on the same 329 ToolMATH-Hard questions. Providing tools does not consistently improve frontier-model accuracy over No tools. Adding distractors increases the number of episodes with tool calls for all three models, while accuracy changes differently across models. The hard split thus enables analysis of tool-use decisions and execution on more difficult problems, without assuming that human-authored tools are indispensable for every question.

Table 5: Same-question controls on all 329 TOOLMATH-HARD questions. Gold-present uses $(\ell = 3, k = 5)$. Accuracy columns are percentages; the last column reports the number of episodes with at least one tool call under Gold-only $\rightarrow$ Gold-present.

Model	No tools Acc.	Gold-only Acc.	Gold-present Acc.	Episodes with calls
Claude Sonnet 4.6	93.92	89.97	86.63	$298 \rightarrow 319$
Gemini 3.1 Pro	93.31	92.40	93.62	$59 \rightarrow 118$
GPT-5	93.92	90.88	91.19	$145 \rightarrow 175$

J ADDITIONAL TRACE-LEVEL FAILURE ANALYSIS

This appendix provides the failure taxonomy, marginal failure distributions, and failure co-occurrence heatmaps used for the trace-level diagnostics summarized in Section 5.6. The taxonomy in Table 6 is non-exclusive: multiple labels may be assigned to a single failed trace. Marginal distributions in Figure 8 show which failure types appear most frequently for each model, while co-occurrence heatmaps show how failure modes are coupled under different tool-catalog conditions.

Table 6: Failure-type taxonomy used in human trace analysis.

Label	Definition
Plan Error (e_1)	The initial plan is ineffective (logically incorrect, or missing essential steps/subgoals).
Tool Selection Error (e_2)	At least one call selects an irrelevant tool or fails to use a suitable tool for a subproblem.
Tool Hallucination (e_3)	At least one call references a tool name that does not exist in the provided tool list.
Wrong Parameter Value (e_4)	At least one call uses schema-valid argument *types* but incorrect/irrelevant values (e.g., wrong constants/inputs, mismatched intermediate values).
Formatting Error (e_5)	At least one call violates the input schema or fails JSON conversion (e.g., wrong types/missing fields, malformed JSON, tool calls in plain text).
Thought Error (e_6)	Reasoning is incorrect given the final answer/tool logic, contradicts observations, or hallucinates `Action`/`Observation` to bypass interaction.
Observation Omission (e_7)	Two or more consecutive `Thought` steps without incorporating an intervening tool `Observation` when a tool interaction is expected.
Repeated Call (e_8)	The exact same *(tool, arguments)* pair is called 2 or more times after being executed (re-calls with different arguments are not counted).
Incomplete Execution (e_9)	No final answer is produced (e.g., timeout, loops/excessive repeats, or exceeding the maximum turn budget).

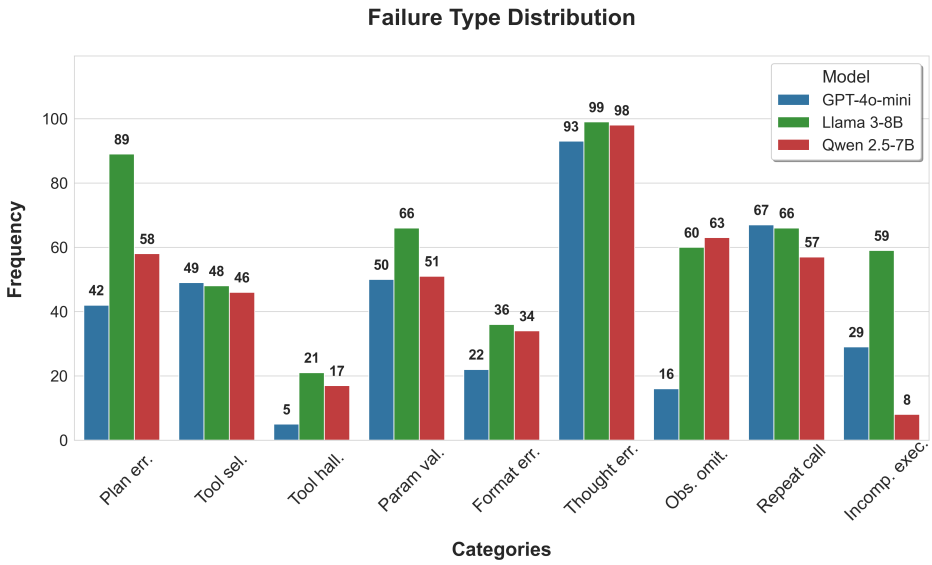


Figure 8: **Failure cases under Gold-present (Level 3, $k = 5$).** Counts are computed from 100 failed instances per model; multiple labels per instance are allowed.

Failure co-occurrence. To analyze coupled failure patterns, we compute error co-occurrence heatmaps. Let e_a and e_b denote failure labels. For each model and tool-catalog condition, we compute

$$H_{a,b} = P(e_a \mid e_b),$$

the fraction of failed traces containing error e_b that also contain error e_a. These conditional statistics summarize which failure modes tend to appear together under each setting and should be interpreted as diagnostic associations rather than causal estimates.

Figure 9 shows that tool-catalog conditions change not only accuracy, but also the structure of model failures. For Llama 3-8B, Tool Selection Error is relatively localized in the Gold-only condition, becomes more broadly coupled with other errors under the Gold-present condition, and is strongly coupled with nearly all other errors under the Distractors-only condition. This suggests that removing gold tools changes tool selection from an isolated mistake into a failure mode that co-occurs with broader execution breakdown.

The same heatmaps reveal model-specific differences under the same catalog condition. In the Gold-only condition, Planning Error is selectively coupled with downstream errors for GPT-4o-mini, more uniformly coupled for Llama 3-8B, and moderately coupled for Qwen 2.5-7B. Thus, TOOLMATH exposes differences in failure organization, not merely differences in final answer accuracy.

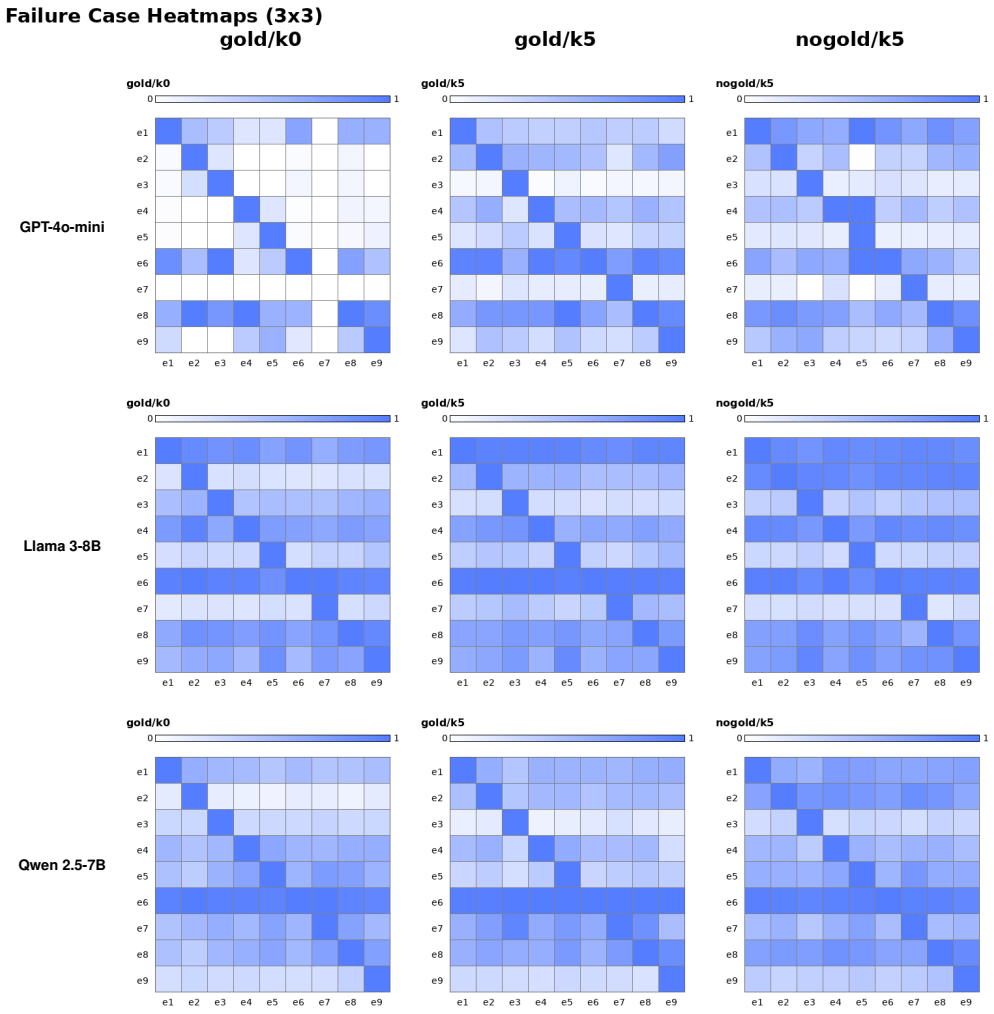


Figure 9: **Failure co-occurrence heatmaps across models and tool-catalog conditions.** Columns correspond to the Gold-only, Gold-present, and Distractors-only conditions. For Gold-present and Distractors-only, we use a distractor budget of $k = 5$ with Level 3 sampling. Rows correspond to GPT-4o-mini, Llama 3-8B, and Qwen 2.5-7B. Each cell reports $P(e_a \mid e_b)$ over failed traces. The labels e_1–e_9 correspond to the failure taxonomy listed in Table 6.

K Detailed Framework Results for ToolMATH-Hard

This appendix reports detailed framework-sensitivity results on ToolMATH-Hard under the Gold-only condition. We compare four execution settings: No tools, ReAct, DFSDT, and Plan+ReAct. The goal is to isolate whether performance differences arise only from the base model, or also from the controller used to organize multi-step tool execution.

Figure 10 summarizes the hop-wise framework comparison across models. At lower hop counts, the differences among ReAct, DFSDT, and Plan+ReAct are often modest, suggesting that short tool-use traces can be handled by relatively simple interaction protocols. At higher hop counts, the gap between controllers becomes clearer: explicit planning tends to maintain accuracy more consistently when later decisions depend on earlier tool observations. DFSDT improves over ReAct in some intermediate-hop regimes, but is less consistently strong than Plan+ReAct at the highest hop depths.

Figures 11, 12, and 13 provide model-specific heatmaps. They show that framework sensitivity is also model-dependent. Llama 3-8B exhibits stronger degradation and larger dependence on the controller, while Qwen 2.5-7B benefits more consistently from Plan+ReAct in higher-hop regimes. These results

support using TOOLMATH-HARD as a diagnostic testbed for controller design in long-horizon tool use.

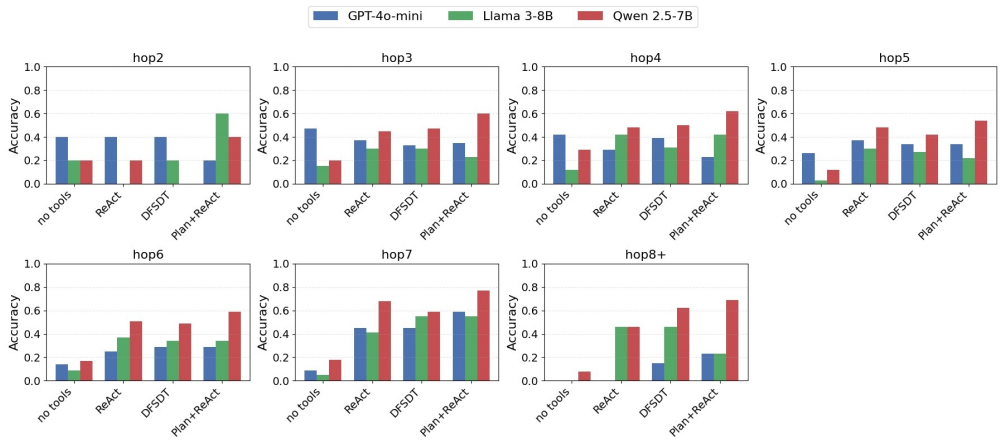


Figure 10: **Hop-wise framework comparison under Gold-only tool lists.** Framework differences are small at low hops but separate at higher hops, where Plan+ReAct most consistently maintains accuracy.

	hop1	hop2	hop3	hop4	hop5	hop6	hop7	hop8+
no tools		0.40	0.47	0.42	0.26	0.14	0.09	0.00
ReAct		0.40	0.37	0.29	0.37	0.25	0.45	0.00
DFSDT		0.40	0.33	0.39	0.34	0.29	0.45	0.15
Plan+ReAct		0.20	0.35	0.23	0.34	0.29	0.59	0.23

Figure 11: Hard-set frameworks for **GPT-4o-mini** (heatmap): no tools vs. ReAct vs. DFSDT vs. Plan+ReAct.

	hop1	hop2	hop3	hop4	hop5	hop6	hop7	hop8+
no tools		0.20	0.15	0.12	0.03	0.09	0.05	0.00
ReAct		0.00	0.30	0.42	0.30	0.37	0.41	0.46
DFSDT		0.20	0.30	0.31	0.27	0.34	0.55	0.46
Plan+ReAct		0.60	0.23	0.42	0.22	0.34	0.55	0.23

Figure 12: Hard-set frameworks for **Llama 3-8B** (heatmap): no tools vs. ReAct vs. DFSDT vs. Plan+ReAct.

	hop1	hop2	hop3	hop4	hop5	hop6	hop7	hop8+
no tools		0.20	0.20	0.29	0.12	0.17	0.18	0.08
ReAct		0.20	0.45	0.48	0.48	0.51	0.68	0.46
DFSDT		0.00	0.47	0.50	0.42	0.49	0.59	0.62
Plan+ReAct		0.40	0.60	0.62	0.54	0.59	0.77	0.69

Figure 13: Hard-set frameworks for **Qwen 2.5-7B** (heatmap): no tools vs. ReAct vs. DFSDT vs. Plan+ReAct.

L Supplementary Split and Catalog Statistics

L.1 Split-wise Logical-Hop Distributions

A shift toward longer reference-solution chains. Table 7 compares the annotated logical-hop distributions. Questions with at least five logical hops account for 74.47% of ToolMath-Hard, versus 40.36% of the main split, a difference of 34.11 percentage points. This establishes a structural difference between the splits; it does not assert that every model must execute that many tool calls or that accuracy decreases monotonically with hop count. Likewise with the main split, models with sufficient internal knowledge may still answer the questions in ToolMath-Hard without calling any tools. Section 5.5 provides the complementary behavioral evidence from frontier-model evaluation.

L.2 Strict-Equivalence Audit and Functional Substitution

How much strict equivalence is detected? An audit of the 12,369-tool pool first groups tools by matching input-type multisets and output types. Within each group, description embeddings from `text-embedding-3-large` identify candidate pairs with cosine similarity at least 0.95; GPT-5 then checks whether the descriptions specify the same semantic operation. Table 8 summarizes the audit. The 797 detected equivalent pairs involve 1,027 tools, or 8.30% of the pool. These alternatives are excluded whenever they are equivalent to a gold tool for the evaluated problem. The 8.30% figure describes tools with a detected equivalent somewhere in the pool; it is not the fraction of distractors in any particular catalog, nor a guarantee that the screen finds every semantic duplicate.

Non-equivalent alternatives can still be useful. Excluding strict equivalents does not remove every way of recovering a gold tool's functionality. A non-equivalent tool may implement one intermediate step, operate on a different representation, or participate in a different solution method. Across the evaluated settings, 5,083 distinct tools (41.09% of the pool) appear as distractors in trajectories that ultimately answer their problems correctly. This is a measure of cross-problem invocation in successful traces, not the percentage of questions solved by substitution or proof that each call was necessary. The pool-level equivalence audit and this behavioral count address different questions and should not be interpreted as a causal comparison.

Appendix H illustrates a concrete form of alternative composition: a model represents complex numbers as two-dimensional vectors and uses vector additions to construct the requested complex arithmetic result. Such a sequence may serve the desired computation without duplicating the gold tool's full operation and typed interface. Retaining these possibilities reflects the mixture of overlap and partial substitutability that motivates the benchmark. Success in Distractors-only can therefore arise from an alternative tool computation or from no-tool reasoning; the decomposition in Section 5.3 separates these observed behaviors before making claims about adaptation.

L.3 Performance-Retention Definitions and Conventions

We next examine how model success changes when the tool catalog is systematically altered. Final accuracy alone does not distinguish whether a model succeeds because it truly uses the intended tools, because it can recover through alternative tool trajectories or internal reasoning, or because it remains

Table 7: Distribution of annotated logical hops in the main and hard splits (percentages). The 5+ row aggregates the preceding 5, 6, and 7+ rows. These percentages describe reference-solution annotations; trajectory-level dependency depth is defined in Section 4.4.1.

Logical hops	TOOLMATH	TOOLMATH-HARD
1	0.38	0.30
2	5.61	1.22
3	21.57	6.08
4	32.08	17.93
5	19.37	27.66
6	13.98	35.56
7+	7.01	11.25
5+ (aggregated)	40.36	74.47

Table 8: Operational strict-equivalence audit of the main 12,369-tool pool. Candidate retrieval is restricted to matching input/output types, followed by a semantic-operation check.

Statistic	Result
Tools in the audit	12,369
Candidate pairs with similarity ≥ 0.95	844
Detected strictly equivalent pairs	797
Tools with at least one detected equivalent	1,027 (8.30%)
Mean number of detected equivalents per tool	0.1289
Maximum number for one tool	10

stable under additional distractors. To capture these tendencies, we define a common *performance retention ratio* (PRR). For each model, distractor similarity level ℓ, and target condition C, PRR is the fraction of examples that are answered correctly in the Gold-only condition and remain correct under condition C:

$$\mathrm{PRR}_\ell^C = \frac{|\{q : \mathrm{Correct}_{\mathrm{Gold}}(q) = 1 \wedge \mathrm{Correct}_{C,\ell}(q) = 1\}|}{|\{q : \mathrm{Correct}_{\mathrm{Gold}}(q) = 1\}|}.$$

Thus, PRR measures the preservation of a model's success after the tool environment changes.

We use this common PRR definition to derive two diagnostics. First, *Adaptability* ($\mathrm{PRR}_1^{\mathrm{DO}}$) measures success retention when the intended tools are removed and only Level 1 distractors remain. It does not identify whether the retained success comes from internal reasoning, alternative tool execution, or both. Level 1 samples different-category tools after equivalence exclusion; it does not guarantee that no useful alternative exists. Second, level-wise *Robustness* ($\mathrm{PRR}_\ell^{\mathrm{GP}}$) measures success retention when non-gold tools are added alongside gold tools (i.e., the Level ℓ Gold-present condition). Higher Robustness indicates less disruption from distractors. We also report the mean and standard deviation of Robustness across Levels 1–5 to summarize overall stability.

Conditioning Adaptability on tool invocation. For model m, let S_m be the questions it answers correctly under Gold-only. Evaluate exactly those questions under the fixed Level 1 Distractors-only configuration, and partition them into S_m^+ (at least one valid executed call) and S_m^0 (no valid executed call). The grouping uses behavior in the *Distractors-only* trajectory, not behavior in the Gold-only trajectory. Writing $Y_m^{\mathrm{DO}}(q)$ for correctness in that condition, define

$$A_m^+ = \frac{\sum_{q \in S_m^+} Y_m^{\mathrm{DO}}(q)}{|S_m^+|}, \qquad A_m^0 = \frac{\sum_{q \in S_m^0} Y_m^{\mathrm{DO}}(q)}{|S_m^0|}, \tag{5}$$

$$r_m = \frac{|S_m^+|}{|S_m|}, \qquad A_m = r_m A_m^+ + (1 - r_m) A_m^0. \tag{6}$$

Thus A_m^+ and A_m^0 are conditional accuracies, whereas $r_m A_m^+$ and $(1 - r_m)A_m^0$ are their contributions to aggregate Adaptability. We omit a conditional accuracy when its denominator is zero; aggregate

Adaptability is undefined if S_m is empty. Every denominator in Table 2 is restricted to the model's Gold-only successes, so these values should not be substituted for full-split Tool-Acc, NoTool-Acc, or call rates in Table 1.

L.4 Additional Catalog-Construction Detail

Input argument names and order do not by themselves distinguish tools; for example, $(\texttt{int}, \texttt{int}, \texttt{str})$ and $(\texttt{str}, \texttt{int}, \texttt{int})$ have the same type multiset. The exclusion criteria and their application before every sampling and fallback stage are detailed in Appendix D.1.

M Scope, Safeguards, and Limitations

The construction pipeline addresses complementary sources of benchmark noise and coverage restriction. Multi-model tool-wise checks test agreement between specifications and execution, question-wise validation tests empirical usability, and targeted human repair revises problematic descriptions or implementations. TOOLMATH-HARD adds a human-authored route for cases unresolved by automatic construction and validation, so the overall benchmark is not restricted to successful trajectories with automatically extracted tools. This mitigates dependence on the automatic pipeline's successes; the unresolved-case selection and the separately validated human tools do not establish complete freedom from model-mediated bias. The methodology constructs controlled instances of connected computation, overlapping capabilities, and missing intended tools; it does not claim to reproduce every property or frequency of deployed API ecosystems.

TOOLMATH is math-grounded, which enables deterministic scoring and controlled construction but does not cover the full ambiguity, underspecification, and open-ended objectives found in general tool-use domains. Its tools are derived from stepwise MATH solutions, so the benchmark emphasizes symbolic tool use rather than interaction with real-world APIs, services, or dynamically changing environments. Although our validation-and-repair pipeline reduces benchmark artifacts, it cannot fully eliminate extraction noise, source-dataset bias, or validation bias. Strict-equivalence screening can miss equivalent tools, and successful calls to non-equivalent distractors do not establish their causal necessity. Invocation-conditioned groups are self-selected, and a verified dependency graph does not by itself show that its deepest component contributed to the final answer. Finally, our behavioral metrics and trace-level co-occurrence analyses are diagnostic rather than causal: they reveal how model behavior changes under controlled tool-catalog shifts, but do not by themselves identify the underlying mechanism responsible for each failure.