NOT EVERY TOOL CALL HELPS: FINE-GRAINED CREDIT ASSIGNMENT FOR EFFICIENT TOOL-INTEGRATED REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-Integrated Reasoning (TIR) enables large language models (LLMs) to solve complex reasoning problems with external tools, yet existing reinforcement learning methods largely rely on coarse-grained trajectory-level rewards and overlook the utility of individual tool calls. We systematically study tool-use behavior and identify pervasive tool overuse: models often invoke tools even when their current reasoning states are already sufficient for correct solutions. To quantify this behavior, we introduce an early-stopping estimator that measures the observable correctness gain of each tool call by comparing outcomes immediately before and after invocation. Our analysis reveals that many tool calls are redundant or harmful, driven by persistent tool-calling bias and limited adaptation to prior tool utility. Based on these findings, we propose GAPO (Gain-Aware Policy Optimization), which converts tool-call gains into fine-grained local advantages and combines them with trajectory-level supervision to optimize tool-calling behavior. Across mathematical reasoning and code-generation benchmarks with Qwen3 models from 4B to 32B, GAPO consistently improves reasoning performance and tool-use efficiency. On Qwen3-8B, it improves average mathematical reasoning accuracy by 13.77 percentage points while reducing tool calls by 80.63%.

1 INTRODUCTION

Large Language Models (LLMs) increasingly rely on external tools such as code interpreters and symbolic executors to solve complex reasoning problems (Mai et al., 2025). Tool-Integrated Reasoning (TIR) extends the reasoning capabilities of LLMs by allowing them to interact with external environments during inference (Lin & Xu, 2025). However, this introduces a new challenge: not every tool call is necessary.

Recognizing this issue, recent work has begun to explicitly regulate tool invocation using signals such as model confidence, knowledge boundaries, or interaction budgets (Li et al., 2025a; Xu et al., 2025a; Liu et al., 2026b). While these methods have achieved moderate improvements in tool-use efficiency, they primarily regulate tool usage at a coarse-grained level, without explicitly characterizing the contribution of individual tool calls. As a result, the fine-grained utility of each invocation, specifically whether it improves, preserves, or degrades the model's ability to solve the problem, remains largely unexplored.

Motivated by this gap, we conduct a systematic study of tool-use behavior. Our preliminary study first reveals a prevalent tool overuse phenomenon: models frequently invoke external tools even when their current reasoning states are already sufficient to solve the problem correctly. To characterize this phenomenon at a finer granularity, we further introduce an early-stopping method to estimate the observable marginal gain of individual tool calls. Based on the resulting correctness transitions, we categorize tool calls into four types—positive-gain, negative-gain, outcome-redundant, and uncertain-gain, and define negative-gain and outcome-redundant calls as overuse calls. Our analysis shows that overuse constitutes a substantial fraction of tool interactions across models and benchmarks. For example, for Qwen3-8B, overuse calls account for 82% and 59% of all tool calls on AMC23 and AIME2024, respectively; at the trajectory level, the model makes 1.54 tool calls on average, while only 0.52 are estimated to be necessary to reach a correct solution. Further analysis

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.

attributes this inefficiency to two complementary factors: a persistent bias toward invoking external tools and a limited ability to adapt subsequent tool-use decisions to the utility of preceding calls. These findings suggest that mitigating tool overuse requires fine-grained supervision that enables models to distinguish beneficial tool calls from redundant or harmful ones and adjust their tool-use strategies accordingly.

Motivated by these findings, we propose **GAPO** (**G**ain-**A**ware **P**olicy **O**ptimization), a turn-level policy optimization framework that directly incorporates the utility of individual tool calls into reinforcement learning. Rather than regulating tool usage through trajectory-level constraints or overall interaction budgets, GAPO provides explicit supervision for each tool-use decision. Specifically, for every tool-call turn, we construct paired early-stopping branches from the states immediately before and after the call, where the model is instructed to produce a final answer without further tool use. Their verifier outcomes determine the gain category of the corresponding tool call, which is then mapped to a category-specific local advantage. GAPO combines this local signal with the conventional trajectory-level outcome advantage and applies it to the native tool-call span, reinforcing beneficial calls while explicitly discouraging redundant or harmful ones. Importantly, the early-stopping branches are used solely to construct local supervision and are excluded from the actor-training batch, allowing GAPO to refine tool-use behavior without altering the original rollout distribution.

We evaluate GAPO on Qwen3 models from 4B to 32B across mathematical reasoning and code-generation benchmarks. Models trained with GAPO consistently improve reasoning performance while substantially reducing tool usage. On Qwen3-8B, GAPO improves average mathematical reasoning accuracy from 43.71% to 57.48% while reducing average tool calls from 1.91 to 0.37. Similar improvements are observed on coding tasks and across model scales, demonstrating that gain-aware supervision enables LLMs to achieve better reasoning performance with substantially fewer external interactions. Overall, our work identifies and characterizes the pervasive tool-overuse problem, introduces an early-stopping framework for quantifying the utility of individual tool calls, and proposes GAPO to translate these fine-grained gain signals into effective policy optimization for more accurate and efficient tool-integrated reasoning.

2 Related Work

2.1 Tool-Augmented Reasoning in Large Language Models

External tools extend LLMs beyond their parametric knowledge and native computational capabilities by providing access to computation, retrieval, and executable environments. Recent work has increasingly integrated tool use directly into the reasoning process. ToolLLM (Qin et al., 2024) and Gorilla (Patil et al., 2024) improve large-scale API selection and invocation, while CodeAct (Wang et al., 2024b) enables multi-turn interaction through executable code actions. More recent Tool-Integrated Reasoning (TIR) methods further interleave internal reasoning with external computation: MathCoder (Wang et al., 2024a) combines natural-language reasoning with code execution, Search-o1 (Li et al., 2025b) incorporates agentic retrieval into long-form reasoning, and ReTool (Feng et al., 2025) uses reinforcement learning with outcome feedback to jointly optimize reasoning and tool-use behavior. Together, these studies demonstrate the effectiveness of external tools in enhancing complex reasoning capabilities.

2.2 Selective and Efficient Tool Use

As tool-augmented reasoning becomes increasingly capable, recent studies have begun to investigate a complementary question: *when should a model invoke a tool?* Unnecessary tool interactions can introduce additional inference cost and latency, and may expose the reasoning process to noisy or erroneous external feedback. Consequently, several approaches explicitly regulate tool invocation rather than treating tool use as an unrestricted component of the reasoning trajectory. MeCo (Li et al., 2025a) introduces metacognitive signals to estimate whether an LLM can solve a problem using its internal capabilities and invokes external tools only when additional assistance is considered necessary. Xu et al. (Xu et al., 2025a) similarly study tool-use decisions from the perspective of knowledge boundaries, using confidence-aware signals to align external assistance with the model's estimated internal competence. Another line of work considers tool usage as a constrained resource

and incorporates interaction budgets into planning or test-time reasoning (Liu et al., 2026b), encouraging models to balance reasoning performance against external interaction cost.

3 PRELIMINARY STUDY

3.1 PROBLEM FORMULATION

We formulate tool-integrated reasoning (TIR) as a partially observable Markov decision process (POMDP) $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega, \mathcal{O} \rangle$, where the agent selects each action according to $a_u \sim \pi_\theta(\cdot \mid h_u)$ based on the observation history h_u. Let τ_i denote the i-th sampled trajectory with N_i tool calls, and let $s_{i,t}$ denote the context state after the first t tool calls, where $t = 0, \ldots, N_i$. Each $s_{i,t}$ contains the original question, the generated reasoning, and all tool observations available up to that point.

The action space is hybrid, $\mathcal{A} = \mathcal{A}_{\text{gen}} \cup \mathcal{A}_{\text{tool}}$. Generation actions $a_u \in \mathcal{A}_{\text{gen}}$ produce tokens for internal reasoning or answer synthesis, whereas tool actions $a_u \in \mathcal{A}_{\text{tool}}$ invoke an external function $c_u = (\textit{func_name}, \textit{args})$. Executing a tool action returns an observation o_{u+1}, which is appended to the history and becomes part of the subsequent reasoning context.

3.2 RLVR ALGORITHMS.

Reinforcement Learning with Verifiable Rewards (RLVR) has emerged as an effective paradigm for improving the reasoning capabilities of large language models, particularly in settings where task outcomes can be automatically verified. By optimizing policies with respect to verifiable outcome-level rewards, RLVR allows models to explore diverse reasoning trajectories and reinforce successful decision-making patterns beyond those directly provided by supervised fine-tuning. In this work, we adopt DAPO Yu et al. (2025) as the underlying reinforcement learning algorithm. DAPO builds upon Group Relative Policy Optimization (GRPO) Guo et al. (2025) and introduces several techniques, including Decoupled Clipping and Dynamic Sampling, to improve optimization stability and maintain effective exploration during long-form reasoning. These properties make it well suited to tool-integrated reasoning, where trajectories may involve multiple interleaved reasoning steps and tool calls. Detailed algorithmic configurations and hyperparameter settings are provided in Appendix E.

4 PRE-STUDY: UNDERSTANDING TOOL OVERUSE

4.1 TOOL OVERUSE IN TOOL-INTEGRATED REASONING

In this section, we examine the tool-use behavior of a broad range of LLMs on reasoning tasks and identify a common phenomenon in Tool-Integrated Reasoning (TIR), which we term *tool overuse*: the invocation of external tools when the model is already capable of solving the problem based on its current internal state.

Settings We first prompt each model to perform eight independent reasoning trials on the benchmark without access to external tools. This no-tool evaluation provides a baseline for assessing each model's intrinsic ability to solve a task. Based on the avg@8 accuracy, we partition the tasks into three difficulty levels: easy ($\text{avg@8} = 1$), medium ($0.5 \leq \text{avg@8} < 1$), and hard ($\text{avg@8} < 0.5$). We then evaluate the same models on the same benchmark under the TIR setting, where external tools are available, again performing eight independent trials. We then group the TIR results according to the above difficulty levels and compare the number of tool calls across groups. We leverage a Python interpreter as a tool, with code execution hosted in a local sandboxed environment to ensure both security and reproducibility. Unless otherwise specified, we use the same tool configuration throughout the remainder of this paper.

Models & Benchmarks The evaluation covers models across two categories: (1) Open-source models: Qwen2.5-7B-Instruct (Yang et al., 2024), Qwen3-8B/32B(Yang et al., 2025). (2) Frontier models: DeepSeek-V3.2(Guo et al., 2025), Kimi-K2.8 (Moonshot AI, 2026), GLM-5.2 (GLM-5 Team et al., 2026). We evaluate all models on AIME2024 and AIME2025.

As demonstrated in Table 1, models exhibit similarly high tool usage across difficulty levels, including on easy tasks that can be solved correctly using only internal knowledge. For example, Qwen3-8B makes an average of 1.9, 2.0, and 2.2 tool calls per task on easy, medium, and hard problems, respectively. This weak variation across difficulty levels suggests that tool invocation is not well aligned with task difficulty, indicating widespread tool overuse in current LLMs.

Table 1: Preliminary study of tool overuse on AIME24/25. We report the average number of tool calls per task.

Model	Easy	Medium	Hard
Open-Source			
Qwen2.5-7B-Inst	0.6	0.6	0.7
Qwen3-8B	1.9	2.0	2.2
Qwen3-32B	3.8	4.1	3.9
Frontier			
DeepSeek-V3.2	3.5	3.7	3.7
Kimi-k2.8	0.4	0.4	0.5
GLM-5.2	1.6	1.7	1.7

4.2 Estimating Tool-Call Gains

While the previous section demonstrates the existence of tool overuse at a macro level, we further investigate this phenomenon by estimating the observable marginal gain of individual tool calls. Specifically, we estimate the observable correctness gain of each tool call by constructing early-stopping branches along the original TIR trajectory. For a trajectory τ_i with N_i tool calls, let $s_{i,t}$ denote the context state after the first t tool calls have been completed, comprising the original question, generated reasoning, and observed tool responses, with $s_{i,0}$ being the state before any tool call. At each stopping point t, we preserve $s_{i,t}$ and append an instruction I_{stop} requiring the model to answer without making further tool calls. This yields an early-stopping branch $B_{i,t} \sim \pi_{\text{old}}(\cdot \mid s_{i,t}, I_{\text{stop}})$ for $t = 0, \ldots, N_i$, where the model may continue reasoning internally but cannot invoke additional tools. Each branch is evaluated by the same answer verifier, giving the correctness label $C_{i,t} = \mathbb{I}[\text{the final answer of } B_{i,t} \text{ is correct}]$.

The observable correctness gain of the t-th tool call is defined as the change in correctness between adjacent branches, where $G_{i,t} = C_{i,t} - C_{i,t-1}$. Accordingly, as shown in Table 2, tool calls are categorized into four types based on their correctness transitions. Based on this taxonomy, we regard both negative-gain and outcome-redundant calls as overuse calls. In both cases, $C_{i,t-1} = 1$, indicating that the preceding context is already sufficient to produce a correct answer without further tool use. The additional tool call therefore provides no positive correctness gain and may even degrade the final outcome. The rationale and empirical validation of this early-stopping branching method are provided in Appendix A, including an ablation on the forced-stop instruction and a repeated-sampling analysis that examines the robustness of gain categorization to the number of sampled early-stopping completions.

Table 2: Categorization of tool calls by observable correctness gain.

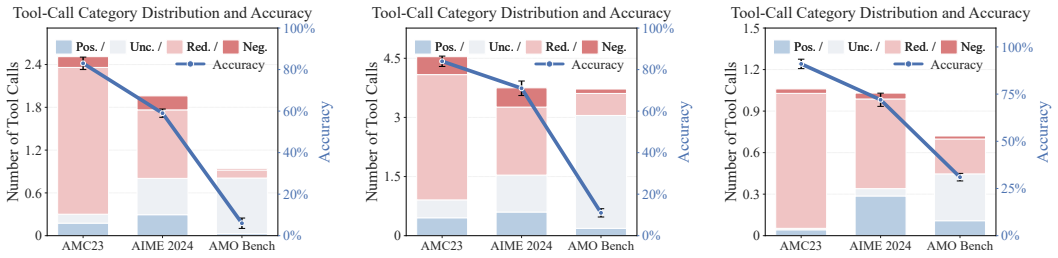
$\mathbf{C}_{i,t-1}$	$\mathbf{C}_{i,t}$	Type
0	1	Positive-gain call
1	0	Negative-gain call
1	1	Outcome-redundant call
0	0	Uncertain-gain call

4.3 Analysis of Tool-Call Gains

Settings Building on the preliminary observations above, we next use the early-stopping estimator in Section 4.2 to characterize tool overuse at both the individual-call and trajectory levels. For each task, we conduct eight independent TIR runs and report results averaged across the eight runs.

Models & Benchmarks We evaluate a diverse set of representative open- and closed-source large language models across mathematics and coding tasks, where answer correctness can be reliably verified. The open-source models include the Qwen3 family at multiple scales (4B, 8B, and 32B(Yang et al., 2025)), while the closed-source models include GLM-5.2 (GLM-5 Team et al., 2026) and Kimi-k2.8 (Moonshot AI, 2026). For mathematics, we consider AMC23 (Art of Problem Solving Foundation, 2023), AIME 2024 (Zhang & Math-AI Team, 2024) and AMO-Bench(An et al., 2025); for coding, we use LiveCodeBench(v6)(Jain et al., 2024).

As shown in Fig. 1, we observe that overuse calls, including both outcome-redundant and negative-gain calls, constitute a substantial fraction of tool invocations. Taking Qwen3-8B as an example, overuse calls account for 82%, 59%, and 14% of all tool calls on AMC23, AIME2024, and AMO-Bench, respectively. Notably, tool overuse becomes more severe as dataset-level accuracy increases,



(a) Qwen3-8B (b) Qwen3-32B (c) GLM-5.2

Figure 1: Distribution of tool-call gain categories across models and benchmarks, together with the corresponding task accuracy. Pos., Unc., Red., and Neg. denote positive, uncertain, outcome-redundant, and negative gain calls, respectively. Overall, overuse calls constitute a substantial fraction of tool interactions, particularly on benchmarks where models achieve higher accuracy.

Table 3: Comparison between the actual and estimated minimum numbers of tool calls. Each entry reports *Actual / Minimum*, where the subscript indicates the reduction relative to the actual tool-call count. AMO, LCB denote AMO-Bench, LiveCodeBench(v6), respectively.

MODEL	AMC23	AIME24	AMO	LCB	AVG.
QWEN3-4B	$0.43/0.00_{\downarrow 0.43}$	$0.31/0.00_{\downarrow 0.31}$	$0.05/0.03_{\downarrow 0.02}$	$1.45/1.23_{\downarrow 0.22}$	$0.56/0.32_{\downarrow 0.24}$
QWEN3-8B	$2.51/0.14_{\downarrow 2.37}$	$1.96/0.51_{\downarrow 1.45}$	$0.94/0.84_{\downarrow 0.10}$	$0.74/0.59_{\downarrow 0.15}$	$1.54/0.52_{\downarrow 1.02}$
QWEN3-32B	$4.53/0.46_{\downarrow 4.07}$	$3.75/0.94_{\downarrow 2.81}$	$3.72/3.23_{\downarrow 0.49}$	$1.43/1.03_{\downarrow 0.40}$	$3.36/1.42_{\downarrow 1.94}$

suggesting that this phenomenon is particularly pronounced on tasks that models can already solve more reliably. Results on other models and code-generation tasks exhibit similar patterns and are provided in Appendix B. Overall, these results demonstrate that tool overuse is prevalent across existing models and task settings. Accordingly, our subsequent method focuses on reducing overuse calls.

To further quantify the extent of tool overuse at the trajectory level, we use the earliest correct early-stopping branch to estimate how many tool calls could have been avoided before the original trajectory terminates. For trajectory τ_i, we define $t_i^{\min} = \min\{t \mid C_{i,t} = 1\}$, the earliest observed stopping point from which the model can already produce a correct answer without invoking additional tools. We then compare this estimated minimum with the actual number of tool calls N_i in the original trajectory, and define the excess tool-call count as $N_i - t_i^{\min}$.

As shown in Table 3, the estimated minimum is substantially lower than the actual number of tool calls across models and datasets. For example, on Qwen3-8B, the model makes 1.54 tool calls on average, whereas only 0.52 calls are estimated to be necessary, corresponding to a 66.2% reduction relative to the observed tool-call count. Similar gaps are consistently observed across other settings. These results indicate substantial room for reducing tool usage without necessarily sacrificing task correctness: models often continue invoking external tools beyond the earliest point at which their accumulated reasoning and tool observations are already sufficient to produce a correct answer.

To further investigate the underlying causes of tool overuse, we conduct additional analyses in Appendix C and identify two complementary factors. First, models exhibit a persistent tool-calling bias, maintaining a strong preference for invoking external tools during reasoning. Second, tool-use decisions adapt poorly to the utility of preceding calls, as the probability that the next call is also an overuse call increases sharply with accumulated overuse (Pearson $r = 0.979$). This persistence suggests that existing reinforcement learning paradigms, which typically rely on coarse-grained, outcome-level rewards, fail to provide models with sufficiently fine-grained signals to distinguish beneficial tool calls from redundant or harmful ones. To address this limitation, we propose GAPO (Gain-Aware Policy Optimization), which leverages observable tool-call gains as turn-level supervision to directly optimize tool-selection behavior.

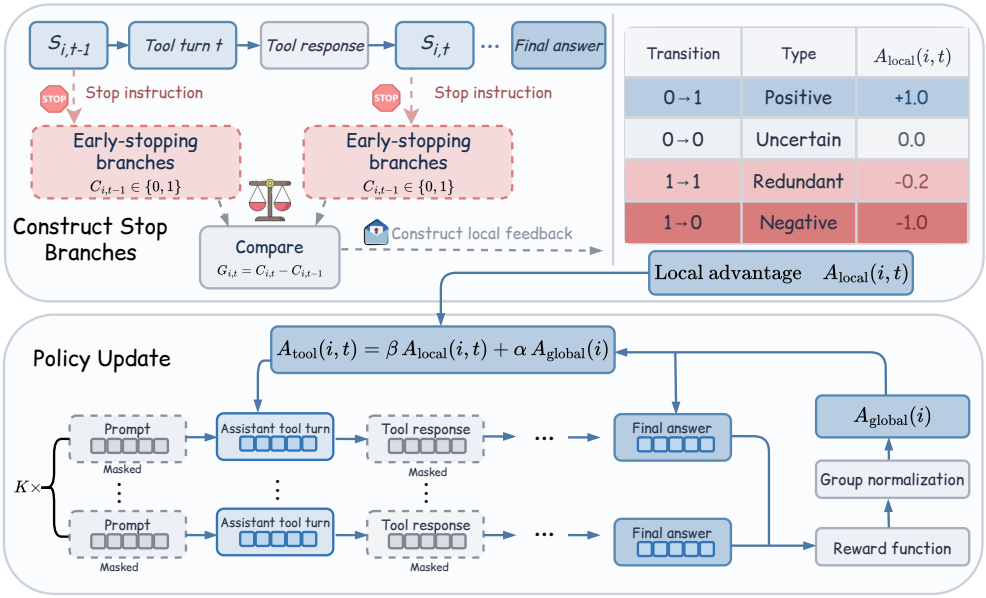


Figure 2: Overview of GAPO. **Top:** For each tool-call turn, GAPO constructs paired early-stopping branches from the two adjacent stopping states. **Bottom:** GAPO performs turn-level policy updates by combining local tool-call advantages with trajectory-level global advantages.

5 METHODOLOGY

Overview. We propose GAPO (Gain-Aware Policy Optimization), a turn-level policy optimization method built upon the DAPO framework (Yu et al., 2025). GAPO retains the group-based RLVR and token-level policy optimization backbone of DAPO, while modifying how policy advantages are constructed and assigned to model-generated tokens. In standard DAPO-style training, a trajectory-level outcome advantage provides a shared supervision signal across the generated response. In contrast, GAPO augments this global signal with fine-grained local supervision derived from the observable tool-call gain estimator introduced in Section 4.2, allowing individual tool-call turns to receive distinct optimization signals. For every question q, GAPO samples a group of K original trajectories $\{\tau_i\}_{i=1}^{K}$ from the rollout policy π_{old}, with each original trajectory retained as a standard actor-training sample. To construct turn-level supervision, GAPO generates paired early-stopping branches before and after each tool call solely for estimating local supervision signals; their tokens are excluded from actor training. This separation allows GAPO to refine the supervision assigned to each original trajectory without changing the rollout distribution used by group-based RLVR.

Local Advantage Computation Following the early-stopping procedure in Section 4.2, for each tool-call turn t of original trajectory τ_i, we construct paired early-stopping branches from the adjacent stopping states $s_{i,t-1}$ and $s_{i,t}$ and evaluate them using the same verifier as the original trajectory. A branch is considered valid if it terminates with a final answer without issuing any additional tool call. For each valid branch, we obtain a binary correctness label $C_{i,t} \in \{0,1\}$ from the verifier outcome. The resulting pair $(C_{i,t-1}, C_{i,t})$ determines the tool-call gain type defined in Section 4.2, and we assign the corresponding fixed local advantage $A_{\text{local}}(i,t)$. Specifically, $A_{\text{pos}}, A_{\text{unc}}, A_{\text{red}}$, and A_{neg} denote the advantage values associated with the positive, uncertain, outcome-redundant, and negative tool-call gain categories, respectively. If either branch is invalid or missing, we set the local advantage to zero. Unlike the global advantage, the local advantage is assigned directly from the paired branch comparison rather than estimated through within-group relative normalization. This preserves the absolute semantics of the observable tool-call gain signal: the same transition receives the same local supervision regardless of the composition or outcome distribution of the sampled trajectory group, while avoiding additional variance from group-level statistics. We additionally study a group-relative variant of the local advantage in Appendix F.

Advantage Assignment. The global advantage is the group-normalized outcome reward of the native trajectory, $A_{\text{global}}(i) = (R_i - \mu_q)/(\sigma_q + \epsilon_{\text{norm}})$, where R_i is the verifier reward for τ_i, and μ_q,σ_q are computed over the K original trajectories for question q. Early-stopping branches are excluded from these statistics. As shown in the lower panel of Figure 2, the two signals are fused at turn level and assigned to token spans:

$$A_{i,j} = \begin{cases} A_{\text{tool}}(i,t) = \beta\, A_{\text{local}}(i,t) + \alpha\, A_{\text{global}}(i), & j \in \mathcal{T}_{i,t}, \\ A_{\text{global}}(i), & j \in \mathcal{F}_i, \\ 0, & \text{otherwise}, \end{cases} \tag{1}$$

where $\alpha, \beta \geq 0$ control the relative contributions of the global and local advantages on tool-call turns, respectively, with $\alpha + \beta = 1$. Here, $\mathcal{T}i,t$ denotes the entire model-generated assistant turn ending with the t-th tool invocation, including both its reasoning and serialized tool-call tokens, while $\mathcal{F}_i$ denotes the entire terminal assistant turn producing the final answer. Together, these spans cover all model-generated tokens in the native trajectory. Prompt, system, tool-response, and padding tokens fall in the last case and are therefore excluded from the policy loss, as in DAPO. An original trajectory that never reaches a terminal answer receives no local stop-gain signal; its global advantage is assigned to its model-generated tokens according to the training configuration.

6 EXPERIMENTS

6.1 SETUP

Implementation details. We apply GAPO to Qwen3-4B/8B/32B(Yang et al., 2025) and refer to the resulting models as GAPO-4B/8B/32B, respectively. For mathematical reasoning, we train on DAPO-Math-17K[1] and evaluate on AMC23 (Art of Problem Solving Foundation, 2023), AIME 24/25 (Zhang & Math-AI Team, 2024; 2025), HMMT 2025(Harvard-MIT Mathematics Tournament, 2025) and AMO-Bench(An et al., 2025), which span a range of reasoning difficulties. For coding, we train on DeepCoder dataset[2] and evaluate on HumanEval(Chen et al., 2021) and LiveCodeBench(v6)(Jain et al., 2024). Unless otherwise specified, all reported evaluation results are averaged over 8 independent runs. Our main experiments use Qwen3-8B with $K = 8$ original trajectories sampled per question. We set the category-specific local advantages to $(A_{\text{pos}}, A_{\text{neg}}, A_{\text{red}}, A_{\text{unc}}) = (1, -1, -0.2, 0)$, with $\alpha = 0.2$ and $\beta = 0.8$ in Eq. equation 1. We allow at most one parallel tool call per tool turn. The remaining optimization settings, including clipping, KL regularization, and batching, are reported in Appendix D.1, while the hyperparameter ranges used in the ablation studies are provided in Section 6.4.

Baselines We compare GAPO with recent RLVR-based methods for tool-integrated reasoning. For mathematical reasoning, our baselines include standard DAPO(Yu et al., 2025), Retool(Feng et al., 2025). Notably, to ensure a consistent comparison, we reimplement Retool on all evaluated base models. For code generation, we compare against DeepCoder(Luo et al., 2025), KodCode(Xu et al., 2025b). To ensure a fair comparison, all coding-related baselines are trained using the same pretrained models as our method and the standard RLVR algorithm.

6.2 MAIN RESULTS

Mathematical Reasoning. As shown in Table 4, GAPO achieves the highest overall accuracy across different model scales and mathematical reasoning benchmarks. In particular, GAPO-8B reaches an average accuracy of **57.48%**, improving over the base model by **13.77** percentage points and outperforming ReTool by **4.56** percentage points. More importantly, this performance gain is accompanied by a substantial reduction in tool usage. Compared with the base model, GAPO-8B reduces the average number of tool calls by **80.63%**, with an especially pronounced reduction of **93.63%** on AMC23. The advantage is particularly evident on Qwen3-4B: ReTool decreases average accuracy from 44.97% to 34.95% while increasing tool usage, whereas GAPO improves accuracy to **52.98%** and further reduces average tool calls from 0.25 to 0.20, highlighting its effectiveness in mitigating tool overuse without sacrificing reasoning performance.

[1] https://huggingface.co/datasets/BytedTsinghua-SIA/DAPO-Math-17k

[2] https://huggingface.co/datasets/agentica-org/DeepCoder-Preview-Dataset

Table 4: Performance comparison across mathematical reasoning benchmarks. Results are reported as accuracy / tool calls. Within each model-size group, the highest accuracy is shown in **bold**, and the lowest available number of tool calls is underlined. AMO denotes AMO-Bench.

MODEL	AMC23	AIME24	AIME25	HMMT25	AMO	AVG.
QWEN3-4B	85.50/0.43	60.48/0.31	49.52/0.13	23.33/0.34	6.00/0.05	44.97/0.25
RETOOL	66.25/10.55	50.00/9.77	33.33/10.93	19.17/10.21	6.00/9.76	34.95/10.24
GAPO-4B	**96.89**/0.07	**66.67**/0.20	**63.33**/0.26	**30.00**/0.28	**8.00**/0.18	**52.98**/0.20
QWEN3-8B	81.25/2.51	58.89/1.96	50.56/2.16	23.33/1.99	4.50/0.94	43.71/1.91
+DAPO	86.25/1.01	65.83/0.87	56.25/0.89	32.50/0.75	10.25/0.65	50.22/0.83
RETOOL	83.43/9.63	70.00/8.36	61.67/10.52	38.75/9.48	10.75/8.31	52.92/9.26
GAPO-8B	**98.75**/0.16	**73.33**/0.32	**62.67**/0.45	**39.17**/0.51	**13.50**/0.42	**57.48**/0.37
QWEN3-32B	82.50/4.53	70.83/3.75	54.17/4.61	25.83/3.99	11.33/3.72	48.93/4.12
RETOOL	84.38/8.01	75.00/10.40	64.17/8.86	33.33/9.13	12.50/8.45	53.88/8.97
GAPO-32B	**97.50**/0.38	**80.00**/0.68	**68.33**/0.59	**35.83**/1.16	**15.50**/1.05	**59.43**/0.77

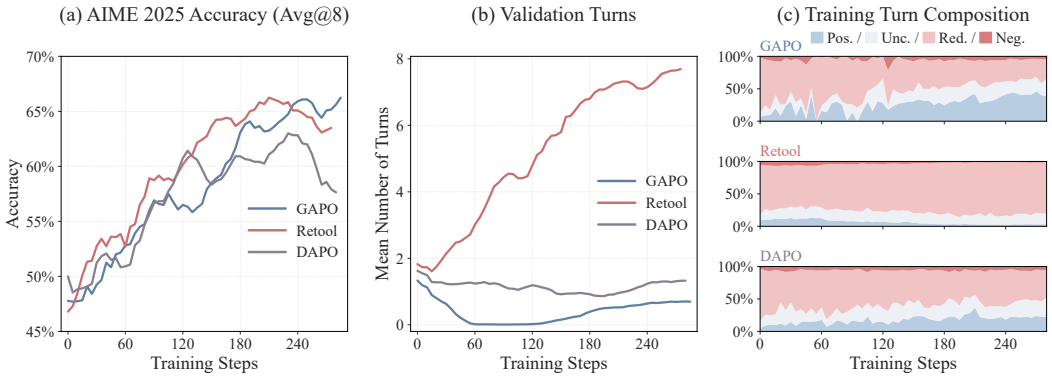


Figure 3: Training dynamics and efficiency comparison. GAPO progressively reduces tool overuse while maintaining strong reasoning performance.

Code Generation. Similar trends are observed on code generation tasks (Table 5). GAPO models achieve competitive or superior performance against code-oriented baselines while using substantially fewer tool calls. In particular, GAPO-8B reaches **90.3%** on HumanEval and **43.3%** on LiveCodeBench, while GAPO-32B achieves the best accuracy on both benchmarks. These results demonstrate that GAPO maintains strong coding performance while improving tool-use efficiency.

Robustness Across Model Scales. GAPO consistently improves reasoning performance while substantially reducing tool usage across 4B, 8B, and 32B models, demonstrating robust scalability across different model capacities.

Table 5: Performance comparison across code generation benchmarks. HE, LCB denote HumanEval, LiveCodeBench(v6), respectively. * denotes models trained on consistent data subsets for fair comparison.

Model	HE	LCB
QWEN3-8B	80.6/0.2	35.2/0.7
DEEPCODER*	**90.6**	39.7
KODCODE	89.3	33.5
GAPO-8B*	90.3/0.1	**43.3**/0.3
QWEN3-32B	90.6/0.3	40.8/1.4
DEEPCODER*	92.4	43.1
GAPO-32B*	**93.3**/0.1	**43.5**/0.9

6.3 TRAINING DYNAMICS AND EFFICIENCY

To investigate how GAPO optimizes the model's tool-calling policy, we analyze the training dynamics of Qwen3-8B, as illustrated in Figure 3. Since the reward function of ReTool explicitly encourages tool invocation, we additionally conduct a standard DAPO (Yu et al., 2025) experiment

using the same reward function as GAPO for a fair comparison. First, as shown in Figure 3(a), GAPO and ReTool achieve comparable accuracy after training stabilizes, both outperforming the DAPO baseline. Second, Figure 3(b) shows that, unlike the steadily increasing tool usage of ReTool, the number of tool calls under GAPO first decreases substantially, followed by a slight rebound from step 120 to step 180. This rebound coincides with a sharp improvement in accuracy and a simultaneous increase in positive-gain calls in Figure 3(c), indicating that GAPO progressively learns to make more effective tool calls. Third, as training progresses, GAPO reduces the proportion of overuse calls from **85.34%** to **30.15%**. In contrast to conventional training strategies such as ReTool, which tend to improve performance at the cost of increased tool overuse, GAPO substantially mitigates overuse while achieving comparable or better reasoning accuracy. We further analyze the total number of model-generated tokens per trajectory and find that GAPO also substantially reduces trajectory length, providing additional evidence of improved inference efficiency; detailed results are reported in the Appendix G.2.

6.4 Ablation Study

We conduct ablation studies using the Qwen3-8B model to isolate the effects of the core GAPO hyperparameters. Specifically, we study the local advantage weight β and the redundant advantage A_{red} in the main text, while additional analysis of the branch sampling size M is provided in the Appendix G.1.

Effect of β. As shown in Figure 4, β controls the balance between trajectory-level supervision and the local tool-call gain signal. The two extreme settings lead to an imbalance between these signals: β=0 reduces to standard DAPO with only trajectory-level supervision, while β=1.0 relies solely on the local gain signal, resulting in either excessive tool use or degraded task performance. The best overall trade-off is achieved at β=0.8, which effectively balances reasoning accuracy and tool-use efficiency.

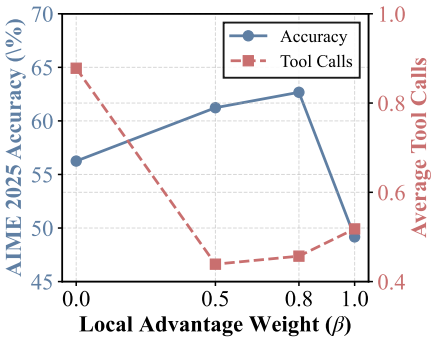


Figure 4: Effect of the local advantage weight β on peak AIME2025 validation accuracy and the corresponding mean tool calls.

Effect of A_{red}. As shown in Table 6, A_{red} controls the penalty assigned to redundant tool calls. A weak penalty provides insufficient pressure to reduce overuse, while overly strong penalties may sacrifice reasoning performance despite further reducing tool usage. Overall, A_{red}=-0.2 achieves the highest average accuracy across the two benchmarks while maintaining a low tool-call frequency, and is therefore adopted as the default setting.

7 Conclusion

In this work, we identify tool overuse as a prevalent issue in tool-integrated reasoning and show that many tool calls are redundant or even harmful to final task performance. To quantify the utility of individual tool calls, we introduce an early-stopping estimator that measures their observable correctness gains. Building on this analysis, we propose GAPO, a gain-aware policy optimization framework that incorporates fine-grained tool-call supervision into reinforcement learning. Experiments across mathematical reasoning and code-generation benchmarks demonstrate that GAPO consistently improves reasoning accuracy while substantially reducing tool usage across different model scales. These results highlight the importance of fine-grained credit assignment for learning more effective and efficient tool-use policies.

Table 6: Ablation on A_{red}. Results are reported as accuracy / tool calls on AIME24 and AIME25.

Setting	AIME24	AIME25
Redundant Advantage A_{red}		
$A_{\text{red}} = -0.1$	72.58/0.50	59.92/0.48
$\mathbf{A_{\text{red}} = -0.2}$	73.33/0.32	**62.67**/0.45
$A_{\text{red}} = -0.3$	**74.21**/0.39	59.76/0.37
$A_{\text{red}} = -0.4$	70.16/0.42	62.50/0.49
$A_{\text{red}} = -0.5$	72.38/0.25	61.75/0.32

LIMITATIONS

Despite promising results, this work presents several limitations that warrant further investigation.

- **Observable Gain as a Partial Measure.** Our gain estimator focuses on the observable correctness gain of individual tool calls, measured by changes in the outcomes of early-stopping branches. However, tool calls may provide other forms of utility that are not directly reflected in immediate answer correctness, such as reducing uncertainty, verifying intermediate reasoning, or providing information that benefits subsequent reasoning. Extending gain estimation beyond correctness-based signals could provide a more comprehensive characterization of tool utility.
- **Limited Tool and Task Scope.** Our experiments primarily focus on mathematically verifiable reasoning and code-generation tasks with a Python interpreter as the external tool. Whether gain-aware supervision generalizes equally well to more diverse tools, such as search engines, APIs, or retrieval systems, and to tasks with less reliable outcome verification remains to be further explored.

AI USE STATEMENT

In this work, we used generative AI tools (GPT5.6-sol) for editing and polishing manuscript text, including grammar correction, phrasing refinement, and structural feedback on drafts. We have not used generative AI tools for generating research ideas, designing experiments, writing code, producing figures or tables, or drafting mathematical proofs, and data collection, model training, and benchmark evaluation are not applicable to AI-assisted generation in this work.

We have reviewed all AI-assisted work. All edited text was manually verified by at least two authors against the original drafts to ensure factual accuracy, logical consistency, and alignment with our experimental evidence. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Shengnan An, Xunliang Cai, Xuezhi Cao, Xiaoyu Li, Yehao Lin, Junlin Liu, Xinxuan Lv, Dan Ma, Xuanlin Wang, Ziwen Wang, et al. Amo-bench: Large language models still struggle in high school math competitions. *arXiv preprint arXiv:2510.26768*, 2025.

Art of Problem Solving Foundation. AMC23: 2023 american mathematics competitions test set. `https://github.com/QwenLM/Qwen2.5-Math/tree/main/evaluation/data/amc23`, 2023. 2023 AMC 12 benchmark.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025.

GLM-5 Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. GLM-5: From vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026. doi: 10.48550/arXiv.2602.15763. URL `https://arxiv.org/abs/2602.15763`.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Harvard-MIT Mathematics Tournament. Harvard-mit mathematics tournament (hmmt) february 2025. `https://www.hmmt.org/www/archive/282`, 2025. Accessed: 2025-02-15.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Aochong Oliver Li and Tanya Goyal. Off-trajectory reasoning: Can llms collaborate on reasoning trajectories? In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/93f3ac5bb6d24ed7cec95ae11209ca34-Abstract-Conference.html.

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*, pp. 3102–3116, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.187. URL https://aclanthology.org/2023.emnlp-main.187/.

Wenjun Li, Dexun Li, Kuicai Dong, Cong Zhang, Hao Zhang, Weiwen Liu, Yasheng Wang, Ruiming Tang, and Yong Liu. Adaptive tool use in large language models with meta-cognition trigger. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13346–13370. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.acl-long.655.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 5420–5438, 2025b. doi: 10.18653/v1/2025.emnlp-main.276.

Heng Lin and Zhongwen Xu. Understanding tool-integrated reasoning. *arXiv preprint arXiv:2508.19201*, 2025.

Boyan Liu, Gongming Zhao, and Hongli Xu. Utility-guided agent orchestration for efficient llm tool use, 2026a. URL https://arxiv.org/abs/2603.19896.

Tengxiao Liu, Zifeng Wang, Jin Miao, I-Hung Hsu, Jun Yan, Jiefeng Chen, Rujun Han, Fangyuan Xu, Yanfei Chen, Ke Jiang, Samira Daruki, Yi Liang, William Yang Wang, Tomas Pfister, and Chen-Yu Lee. Budget-aware tool-use enables effective agent scaling. In *Third Conference on Language Modeling*, 2026b. URL https://arxiv.org/abs/2511.17006.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepcoder: A fully open-source 14b coder at o3-mini level. https://pretty-radio-b75.notion.site/DeepCoder-A-Fully-Open-Source-14B-Coder-at-O3-mini-Level-1cf81902c14680b3bee5eb349a512a51, 2025. Notion Blog.

Xinji Mai, Haotian Xu, Xing W, Weinong Wang, Yingying Zhang, and Wenqiang Zhang. Agentic rl scaling law: Spontaneous code execution for mathematical problem solving. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0251.

Moonshot AI. Kimi k2.8 preview. Kimi Code Documentation, 2026. URL https://www.kimi.com/code/docs/en/kimi-code/whats-new.html. Released September 11, 2026.

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*, volume 37, 2024.

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.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning. In *International Conference on Learning Representations*, 2024a.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232, 2024b.

Hongshen Xu, Zihan Wang, Zichen Zhu, Lei Pan, Xingyu Chen, Shuai Fan, Lu Chen, and Kai Yu. Alignment for efficient tool calling of large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17776–17792. Association for Computational Linguistics, 2025a. doi: 10.18653/v1/2025.emnlp-main.898.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, 2025b.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Yirong Zeng, Shen You, Yufei Liu, Qunyao Du, Xiao Ding, Yutai Hou, Yuxian Wang, Wu Ning, Haonan Song, Dandan Tu, Bibo Cai, and Ting Liu. The tool-overuse illusion: Why does llm prefer external tools over internal knowledge?, 2026. URL https://arxiv.org/abs/2604.19749.

Yifan Zhang and Math-AI Team. American invitational mathematics examination (aime) 2024, 2024. URL https://huggingface.co/datasets/math-ai/aime24.

Yifan Zhang and Math-AI Team. American invitational mathematics examination (aime) 2025, 2025. URL https://huggingface.co/datasets/math-ai/aime25.

A Details of Early-Stopping Evaluation and Tool Call Categorization

A.1 Forced-stop instruction

Stop Instruction Formulation. For each early-stopping branch, we append the following instruction to the preserved context:

```
[FORCED_STOP_BRANCH] This is a counterfactual stop decision.  Answer
the original user question now, using only the information already
available in the conversation.  You MUST provide your best complete
final answer in this turn.  You MUST NOT call any tool.  You MUST NOT
emit <tool_call> or </tool_call>.  You MUST NOT ask to continue or
state that more tool use is required.  Even if some information is
```

Table 7: Consistency between native trajectories and final-answer branches with the forced-stopping instruction.

Dataset	Correctness-label agreement
AMC23	96.14
AIME2024	92.16
AIME2025	94.62
Average	**94.31**

```
uncertain, make the best possible answer now.  The final answer must
follow the required \boxed{...} format.  [/FORCED_STOP_BRANCH]
```

Instruction-interference ablation. To examine whether the stop instruction itself interferes with answer generation, we construct an ablation branch only at the end of each native trajectory. More precisely, for a trajectory τ_i with N_i tool calls, we evaluate the branch at s_{i,N_i}, after all tool responses have already been observed. Thus, the branch and the native trajectory contain the same question, reasoning history, and tool responses; the only intervention at the final answering stage is the appended instruction I_{stop}. Both outputs are evaluated using the same answer verifier. We conduct this ablation using Qwen3-8B on AMC23, AIME24, and AIME25. Each configuration is repeated 8 times with independent sampling, and we report the average agreement across runs. Let C_i^{native} denote the correctness label of the native trajectory's final answer, and let C_{i,N_i}^{stop} denote the label of the corresponding final-answer branch. We measure their consistency as

$$\text{Agreement} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}\Big[C_i^{\text{native}} = C_{i,N_i}^{\text{stop}}\Big] .$$

As shown in Table 7, the correctness labels of the native trajectories and the final-answer branches agree in approximately 94.31% of cases. This high agreement indicates that the stop instruction does not materially alter the model's answer correctness, supporting its use for estimating the observable gains of individual tool calls. The remaining disagreement may result from generation stochasticity or from the instruction changing the model's final-answer behavior in a small number of cases.

A.2 Interpretation of Observable Gains

Interpretation of gain types. A positive-gain call changes the early-stopping outcome from incorrect to correct, indicating an observable improvement in the model's ability to solve the problem after the tool call. Conversely, a negative-gain call changes the outcome from correct to incorrect, suggesting that the state transition induced by the call may adversely affect the model's subsequent reasoning or answer generation. An outcome-redundant call leaves the answer correct both before and after the call, indicating that the call produces no additional observable correctness gain under the early-stopping evaluation. Finally, an uncertain-gain call corresponds to the case where both adjacent branches are incorrect. Such a call yields no immediate observable correctness improvement, but it may still serve as a necessary intermediate step in a multi-step tool-use trajectory.

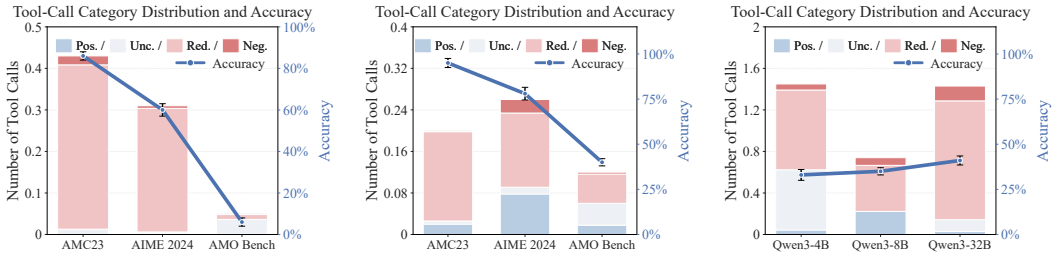
Scope of the categorization. Importantly, this categorization is defined with respect to the correctness outcomes of the early-stopping branches. In particular, "outcome redundancy" only means that no additional correctness gain is observed under our evaluation; it does not imply that the tool call has no effect on other properties of the model's behavior, such as answer confidence, robustness, or reasoning stability. Moreover, because both the stopping instruction and stochastic generation may influence branch outcomes, $G_{i,t}$ should be interpreted as a proxy for the observable state-transition gain associated with a tool call, rather than as an exact estimate of its true expected utility.

A.3 Effect of Early-Stopping Sampling Frequency

Single-sample versus repeated branch evaluation. Our observable tool-call gain estimator uses one sampled completion for each early-stopping branch. Since branch generation is stochastic,

Table 8: Effect of the number of samples per early-stopping branch. We report the agreement of tool-call gain categories with repeated-sampling estimates.

# Samples	Agreement
2	96.5%
4	93.7%
8	94.4%



(a) Qwen3-4B (b) Kimi-K2.8 (c) Code

Figure 5: Additional tool-call category distributions and dataset-level accuracy.

we examine whether repeated sampling would materially change the resulting tool-call categorization. Specifically, we conduct an ablation with Qwen3-8B on AIME24 and AIME25, independently sampling each early-stopping branch 2, 4, or 8 times under the same forced-stop instruction and decoding configuration, and compare the resulting gain categories with those obtained using a single sample. As shown in Table 8, the gain-category assignments remain highly consistent with the single-sample setting, with agreement rates of 96.5%, 93.7%, and 94.4% for 2, 4, and 8 samples, respectively. Despite minor fluctuations across sampling counts, all settings achieve over 93% agreement, suggesting that repeated branch sampling introduces only limited changes to the resulting tool-call supervision. Based on this observation, we use a single completion for each early-stopping branch throughout our experiments. This substantially reduces the additional rollout and verification cost while preserving largely consistent tool-call gain assignments.

B Additional Analysis of Tool-Call Gains

We provide additional results on different models and code-generation tasks to further examine the prevalence of tool overuse. Figure 5 reports the distribution of tool-call categories together with the corresponding dataset-level accuracy.

Results on additional models. Figures 5(a) and 5(b) show the results for Qwen3-4B and Kimi-K2.8, respectively. The observations are broadly consistent with those reported in the main text: outcome-redundant and negative-gain calls constitute a substantial portion of tool invocations. These results provide additional evidence that tool overuse is especially pronounced on problems that the underlying model is already more capable of solving.

Results on code-generation tasks. Figure 5(c) further shows that tool overuse is not specific to mathematical reasoning. Across Qwen3-4B, Qwen3-8B, and Qwen3-32B, redundant and negative-gain calls consistently account for a considerable fraction of tool invocations on the code-generation task. Notably, the overuse fraction remains large as model scale increases and becomes particularly pronounced for Qwen3-32B, whose tool calls are overwhelmingly categorized as redundant or negative despite its relatively higher task accuracy.

Overall, the additional results demonstrate that tool overuse is robust across model families, model scales, and task domains.

C Detailed analysis of overuse

C.1 Intrinsic Tool-Calling Bias.

Settings To investigate whether tool overuse originates from an intrinsic preference for tool invocation, we explicitly constrain tool usage through the system prompt and examine whether models can suppress tool calls under increasingly restrictive instructions. Specifically, we consider two levels of restriction. Under the *tool-discouraged* setting, models are instructed to rely on their own reasoning whenever possible and to invoke tools only when the problem cannot be answered reliably without external assistance. Under the more restrictive *tool-prohibited* setting, models are explicitly instructed not to invoke any tool under any circumstances and to answer exclusively based on their internal reasoning and knowledge. We evaluate both settings on AIME24/25 using the Qwen3-8B model, with the same inference configuration as in the main experiments.

Prompt The system prompt used for the *tool-discouraged* setting is shown below:

```
You are a helpful assistant. # Tools You may have access to one or
more functions that can assist with the user query. You are provided
with function signatures within <tools></tools> XML tags: <tools>
{"type": "function", "function": {"name": "code_interpreter",
"description": "A tool for executing code.", "parameters": {"type":
"object", "properties": {"code": {"type": "string", "description":
"The code to execute."}}, "required": ["code"]}}} </tools> Tool
usage is generally discouraged. Prefer answering the user's query
directly using your own reasoning and knowledge whenever possible.
Only call a tool when the query cannot be answered reliably without
using the provided tools. Do not call tools merely because relevant
tools are available. If you decide to call a function, return a json
object with function name and arguments within <tool_call></tool_call>
XML tags: <tool_call> {"name": <function-name>, "arguments":
<args-json-object>} </tool_call> The answer format must be:
\boxed{{'The final answer goes here.'}}
```

For the *tool-prohibited* setting, we further strengthen the constraint by explicitly forbidding any tool invocation:

```
You are a helpful assistant. # Tools You may have access to one or
more functions that can assist with the user query. You are provided
with function signatures within <tools></tools> XML tags: <tools>
{"type": "function", "function": {"name": "code_interpreter",
"description": "A tool for executing code.", "parameters": {"type":
"object", "properties": {"code": {"type": "string", "description":
"The code to execute."}}, "required": ["code"]}}} </tools> You must
not call or use any tools under any circumstances. Do not output
<tool_call> tags or attempt to invoke any function, even if a tool
would help answer the query. Answer the user's query using only your
own reasoning and knowledge. The answer format must be: \boxed{{'The
final answer goes here.'}}
```

As shown in Table 9, explicit restrictions reduce but do not eliminate tool invocation. On AIME24/25, the model still makes an average of 1.96 and 0.34 tool calls per trajectory under the *tool-discouraged* and *tool-prohibited* settings, respectively. This persistence under increasingly restrictive instructions suggests an intrinsic tool-calling bias that contributes to tool overuse. This observation is consistent with prior work. (Zeng et al., 2026) attribute such excessive tool-use tendencies to inaccurate perceptions of models' knowledge boundaries and prevailing tool-use training paradigms, where models are primarily trained to execute valid tool calls rather than to assess their actual necessity or utility (Qin et al., 2024; Li et al., 2023).

Table 9: Tool-call frequency under different levels of explicit tool-use restriction on AIME24 and AIME25. We report the average number of tool calls per trajectory.

Setting	AIME24	AIME25	Avg.
Tool-Discouraged	2.01	1.91	1.96
Tool-Prohibited	0.22	0.45	0.34

Table 10: Tool-use behavior under different levels of accumulated overuse.

Prior Overuse Ratio	P_{call}	$P_{\text{next-overuse}}$
0%	62.7%	20.1%
$(0\%, 20\%]$	73.4%	43.4%
$(20\%, 40\%]$	65.0%	57.6%
$(40\%, 60\%]$	61.4%	72.0%
$(60\%, 80\%]$	58.6%	83.3%
$(80\%, 100\%]$	59.7%	97.6%
Pearson r	-0.682	0.979

C.2 Limited Tool-Utility Estimation and Dynamic Adaptation.

Settings We further investigate whether models can estimate the utility of tool calls and dynamically adjust their tool-use behavior throughout the reasoning process. We conduct further analysis on the Qwen3-8B results in Section 4.3. For each reasoning state $s_{i,t}$, we compute the proportion of overuse calls among all preceding evaluable tool calls as $R_{i,t} = \frac{1}{N^{\text{eval}}_{i,<t}} \sum_{j<t} \mathbb{I}[(C_{i,j-1}, C_{i,j}) \in \{(1,1),(1,0)\}]$. We then measure (1) the probability of invoking another tool, $P_{\text{call}}(r) = \Pr(\text{tool call} \mid R_{i,t} = r)$, and (2) conditioned on another tool call being made, the probability that it is again an overuse call, $P_{\text{next-overuse}}(r) = \Pr((C_{i,t}, C_{i,t+1}) \in \{(1,1),(1,0)\} \mid \text{tool call}, R_{i,t} = r)$. In practice, we group $R_{i,t}$ into intervals and estimate both probabilities by averaging within each interval.

The results reveal a clear mismatch between tool-use probability and realized tool utility, indicating that the model does not effectively adapt its tool-use policy to accumulated evidence of low utility. Even when almost all previous tool calls are already overuse calls, the model still maintains a tool-calling probability comparable to that observed when no overuse has accumulated (59.7% vs. 62.7%). Meanwhile, the probability that the next tool call is again an overuse call increases sharply with the accumulated overuse ratio, reaching 97.6% when the historical ratio falls in $80\% - 100\%$, with a Pearson correlation of 0.979, thereby reinforcing tool overuse. This observation is consistent with prior work. (Liu et al., 2026a) report that tool-call utility can decline substantially as reasoning progresses while models continue to invoke tools with limited adaptation. (Zeng et al., 2026) attribute this behavior partly to the *outcome reward trap*, where trajectory-level rewards provide insufficient supervision for distinguishing the utility of individual tool calls. (Li & Goyal, 2026) further suggests that models may mistake additional context or newly retrieved information for genuine task progress, leading to repeated tool use even when further calls provide little benefit.

D Details in GAPO

D.1 Training Hyperparameter Settings

We summarize the hyperparameters used for training GAPO in Table 11. The configuration includes the general training setup, DAPO optimization parameters, and GAPO-specific coefficients. Unless otherwise specified, the same hyperparameter configuration is applied across all model scales evaluated in our experiments.

Table 11: Training hyperparameter settings for GAPO.

Configuration	Value
General Training	
Global Batch Size	64
Mini-batch Size	8
Rollouts per Prompt	8
Max Prompt Length	2,048
Max Response Length	16,384
Actor Learning Rate	1×10^{-6}
Sequence Parallel Size	8
DAPO Clipping	
Clipping Ratio Low (ε_{low})	0.2
Clipping Ratio High($\varepsilon_{\text{high}}$)	0.28
Clipping Ratio C	10.0
Global Advantage Weight	
Local Advantage Weight (β)	0.8
Global Advantage Weight (α)	0.2
GAPO Local Advantages	
Positive-gain Advantage (A_{pos})	1.0
Negative-gain Advantage (A_{neg})	-1.0
Outcome-redundant Advantage (A_{red})	-0.2
Uncertain-gain Advantage (A_{unc})	0.0

D.2 Inference Settings

For evaluation, we use the final trained checkpoint of each model and adopt a consistent inference configuration across all experiments. The decoding and context-length settings are summarized in Table 12. Unless otherwise stated, these settings are shared across all evaluated model scales and benchmarks.

Table 12: Inference settings used for GAPO evaluation.

Configuration	Value
Temperature (τ)	1.0
Top-p	0.6
Max Prompt Length	2,048
Max Generation Length	16,384
Tensor Parallel Size	4
Max Tool Turns	16

D.3 Hardware Configuration

All training and evaluation experiments were conducted on a cluster of **64 Ascend 910B NPUs** distributed across **8 compute nodes**. For training, sequence parallelism with a parallel size of 8 was employed to support long-context optimization. During inference, tensor parallelism with a parallel size of 4 was used to accommodate the memory requirements of long-context generation.

Under this hardware configuration, each RL training run for the 4B- and 8B-scale models was completed within approximately 72 hours, while training the 32B-scale models required approximately 96 hours. The same hardware platform was used for evaluation to ensure a consistent experimental setup across model scales.

E Detailed RLVR Algorithms

Group Relative Policy Optimization (Shao et al., 2024) efficient optimize the model's reasoning capabilities without the significant memory overhead of a Value Network. Unlike Proximal Policy Optimization Schulman et al. (2017), which relies on a separate value function to estimate the baseline, GRPO leverages the distribution of rewards within a group of sampled outputs to compute the advantage.

Group Sampling and Advantage Estimation. For each query q, GRPO samples a group of G outputs $\{o_1, o_2, \ldots, o_G\}$ from the current policy π_θ. Let r_i be the reward for the i-th output. The advantage A_i is computed by normalizing the reward against the group statistics:

$$A_i = \frac{r_i - \bar{r}}{\sigma_r + \epsilon} \tag{2}$$

where $\bar{r}$ and σ_r are the mean and standard deviation of the rewards within the group, respectively. This group-based baseline serves as a dynamic reference, reducing variance without requiring a learned value function.

Objective Function. The optimization objective maximizes the surrogate loss with a KL-divergence penalty to maintain stability relative to the reference model π_{ref}:

$$\mathcal{J}(\theta) = \mathbb{E}_{q\sim\mathcal{D},\{o_i\}\sim\pi_{\theta_{\text{old}}}} \left[\frac{1}{G} \sum_{i=1}^{G} \left(\mathcal{L}_{\text{clip}}(r_t(\theta), A_i) - \beta \mathbb{D}_{\text{KL}}(\pi_\theta || \pi_{\text{ref}}) \right) \right] \tag{3}$$

Here, $r_t(\theta) = \frac{\pi_\theta(o_i|q)}{\pi_{\theta_{\text{old}}}(o_i|q)}$ denotes the probability ratio, and $\mathcal{L}_{\text{clip}}$ is the standard PPO clipping operator:

$$\mathcal{L}_{\text{clip}}(r_t, A_i) = \min\left(r_t A_i, \text{clip}(r_t, 1-\epsilon, 1+\epsilon) A_i\right) \tag{4}$$

By eliminating the critic model, GRPO significantly reduces computational costs, enabling efficient training of large-scale reasoning models.

Decoupled Clip and Dynamic sAmpling Policy Optimization (DAPO). While GRPO offers efficiency, it often suffers from entropy collapse and training instability when dealing with complex reasoning and tool-call sequences. To address this, we employ DAPO as the RLVR algorithm in this paper. It introduces four key techniques to make RL shine in the long-CoT RL scenario: (1) Clip-Higher, which promotes the diversity of the system and avoids entropy collapse; (2) Dynamic Sampling, which improves training efficiency and stability; (3) Token-Level Policy Gradient Loss, which is critical in long-CoT RL scenarios; (4) Overlong Reward Shaping, which reduces reward noise and stabilizes training.

Totally, DAPO samples a group of outputs $\{o_i\}_{i=1}^{G}$ for each question q paired with the answer a, and optimizes the policy via the following objective:

$$\begin{aligned}
\mathcal{J}_{\text{DAPO}}(\theta) =& \mathbb{E}_{(q,a)\sim\mathcal{D},\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \\
& \left[\frac{1}{\sum_{i=1}^{G}|o_i|} \sum_{i=1}^{G} \sum_{t=1}^{|o_i|} \min\left(r_{i,t}(\theta)\hat{A}_{i,t},\ \text{clip}(r_{i,t}(\theta), 1-\varepsilon_{\text{low}}, 1+\varepsilon_{\text{high}})\hat{A}_{i,t} \right) \right] \\
& \text{s.t.} \quad 0 < \left| \{o_i \mid \text{is_equivalent}(a, o_i)\} \right| < G,
\end{aligned} \tag{5}$$

where

$$r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})}, \quad \hat{A}_{i,t} = \frac{R_i - \text{mean}(\{R_i\}_{i=1}^{G})}{\text{std}(\{R_i\}_{i=1}^{G})}. \tag{6}$$

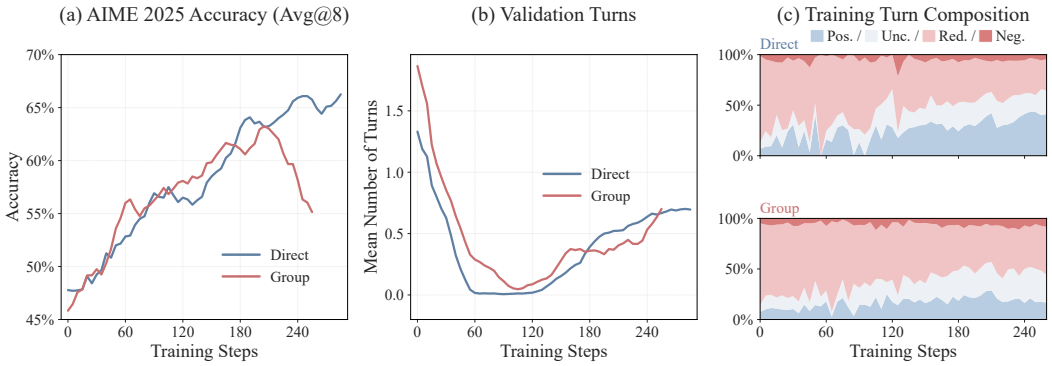


Figure 6: **Ablation of direct versus group-relative local advantage assignment.** *Direct* denotes the proposed GAPO formulation, which assigns the fixed gain-type advantage directly to each tool-call transition; *Group* instead normalizes local advantages within the trajectory group for each question. Direct assignment yields more stable late-stage optimization and a progressively larger fraction of positive tool-call transitions.

F Group-Relative Local Advantage.

We additionally study a group-relative variant of the local advantage to examine whether the observable tool-call gain signal benefits from within-group normalization. In the proposed GAPO formulation (*Direct*), each valid tool-call transition is assigned the fixed advantage associated with its observed gain type, independent of the other trajectories sampled for the same question. As an ablation (*Group*), we instead normalize these local advantages within the trajectory group for each question. Specifically, letting μ_q^{local} and σ_q^{local} denote the mean and standard deviation of the valid local advantages collected from the K native trajectories for question q, we replace $A_{\text{local}}(i,t)$ with

$$\widetilde{A}_{\text{local}}(i,t) = \frac{A_{\text{local}}(i,t) - \mu_q^{\text{local}}}{\sigma_q^{\text{local}} + \epsilon_{\text{norm}}}. \tag{7}$$

The resulting $\widetilde{A}_{\text{local}}(i,t)$ is then combined with the global advantage using the same turn-level objective as in Eq. 1; all other training settings are kept unchanged.

As shown in Figure 6, the group-relative variant tracks the direct assignment during the early and middle stages of training, but becomes substantially less stable later in training. While *Direct* continues improving and reaches 66% AIME 2025 Avg@8 accuracy, *Group* peaks earlier and subsequently degrades to 55%. This difference is not explained by a qualitatively different tool-use budget: the mean number of validation turns follows a similar trajectory for the two variants, first decreasing toward zero and then gradually increasing again as training proceeds. Instead, their training-turn compositions diverge markedly. Under direct assignment, the fraction of positive tool-call transitions progressively increases over training, whereas group-relative normalization maintains a substantially larger fraction of uncertain and outcome-redundant turns and a smaller fraction of positive turns. These results suggest that normalizing the local signal within each sampled group weakens its absolute transition-level semantics: the supervision assigned to the same observable gain type becomes dependent on the composition of the other sampled trajectories. Directly assigning the gain-type advantage avoids this coupling and provides a more stable local supervision signal.

G Additional Ablations and Efficiency Analysis

G.1 Effect of the Branch Sampling Size M.

We study the effect of the number of early-stopping branches sampled from each stopping state, denoted by M. When $M > 1$, the binary branch outcomes are replaced by their mean correctness at each stopping state, and the local gain is determined by comparing these averaged outcomes. As shown in Table 13, different values of M yield comparable reasoning performance and tool-use

Table 13: Ablation on the branch sampling size M. Results are reported as accuracy / tool calls on AIME24 and AIME25.

Setting	AIME24	AIME25
$M = 1$	73.33/0.32	**62.67**/0.45
$M = 2$	**75.01**/0.34	60.35/<u>0.42</u>
$M = 4$	72.57/0.41	61.24/0.49

Table 14: Average number of model-generated tokens per trajectory. Lower values indicate shorter and more efficient reasoning trajectories. The lowest value in each column is <u>underlined</u>. All models are based on Qwen3-8B.

METHOD	AMC23	AIME24	AIME25	HMMT25	AMO-BENCH	AVG.
QWEN3-8B	6.44K	10.26K	11.30K	12.89K	14.46K	11.07K
+DAPO	7.27K	10.78K	11.80K	13.39K	15.59K	11.76K
RETOOL	7.89K	12.60K	14.40K	15.99K	16.71K	13.52K
GAPO-8B	<u>4.59K</u>	<u>8.04K</u>	<u>9.24K</u>	<u>8.86K</u>	<u>10.08K</u>	<u>8.16K</u>

efficiency. Since increasing M incurs additional branch-generation and verification costs without consistent gains, we set M=1 by default for efficiency.

G.2 MODEL-GENERATED TOKENS PER TRAJECTORY

Beyond tool-call frequency, we further measure the total number of model-generated tokens per trajectory to evaluate inference efficiency. As shown in Table 14, GAPO consistently produces the shortest trajectories across all five benchmarks. On average, GAPO reduces the number of generated tokens from 11.07K for Qwen3-8B, 11.76K for DAPO, and 13.52K for ReTool to only 8.16K, corresponding to reductions of approximately 26%, 31%, and 40%, respectively. This consistent reduction suggests that GAPO does not mitigate tool overuse by simply shifting computation toward longer internal reasoning, but instead yields more concise and efficient reasoning trajectories overall.