FLRT: LEARNING EFFICIENT TOOL USE FROM TOOL-SUFFICIENT STATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-augmented reasoning substantially extends the capability of large language models to solve complex problems, but frequent tool use also incurs additional interaction and execution costs. We observe that tool necessity changes dynamically along a reasoning trajectory: a model may rely on external tools to acquire critical information early on, yet become capable of completing the task independently at later states. To characterize this transition, we introduce Fork Exploration, which disables future tool access from real intermediate states and observes whether the model can still reach a correct solution, yielding a state-level signal of tool sufficiency. Building on this signal, we propose FLRT, a two-stage training framework that learns more efficient tool-use policies through fork-guided supervision and fork-aware reinforcement learning. Experiments across three mathematical reasoning benchmarks show that FLRT improves average accuracy over the base model while reducing tool calls by about one-third. Further analysis shows that FLRT primarily reduces tool interactions after the model has already acquired sufficient information, suggesting that explicitly modeling within-trajectory changes in tool necessity is an effective approach to improving the efficiency of tool-augmented reasoning.

1 INTRODUCTION

Large language models have made steady progress on mathematical reasoning through step-by-step prompting and reinforcement learning (Wei et al., 2022; DeepSeek-AI et al., 2025). Tool-augmented methods interleave natural-language reasoning with external computation: a model issues a request, an interpreter returns its result, and the model incorporates that information into subsequent reasoning (Gou et al., 2023; Yao et al., 2022).

However, greater capability brings additional costs. Each call requires generating a request, executing a program, waiting for its response, and extending the context. Serial interactions can increase latency and context overhead. Efficient tool use therefore requires maintaining solution quality with fewer external interactions. Prior work studies this trade-off through correctness–efficiency rewards (Wang et al., 2025a), prompting and tool-need prediction (Sun et al., 2026), and supervision of tool decisions at individual subgoals (Qian et al., 2025).

However, optimizing the total call count alone treats calls as uniform costs, while tool necessity changes along a reasoning trajectory. The available information includes the question, generated reasoning, previous calls, and their returned results. A problem may require external computation initially, yet become solvable without further tools after critical results enter the context. Calls after that point have a different role from earlier information acquisition. A trajectory-level count does not locate this transition or distinguish premature stopping from continued interaction after the model can already finish.

We ask: **when can a model complete a tool-augmented reasoning trajectory correctly using only its current context?** To investigate this question, **Fork Exploration** restores a checkpoint before a tool call in a correct trajectory, preserves the full observed prefix, disables all future tools, and generates a new continuation. A successful continuation provides an observed path to the answer without additional tool use. All 25 correct trajectories in our 30-problem AIME25 pilot contain such a state. We characterize the corresponding continued interaction in the original trajectory as **Tool Redundancy**.

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.

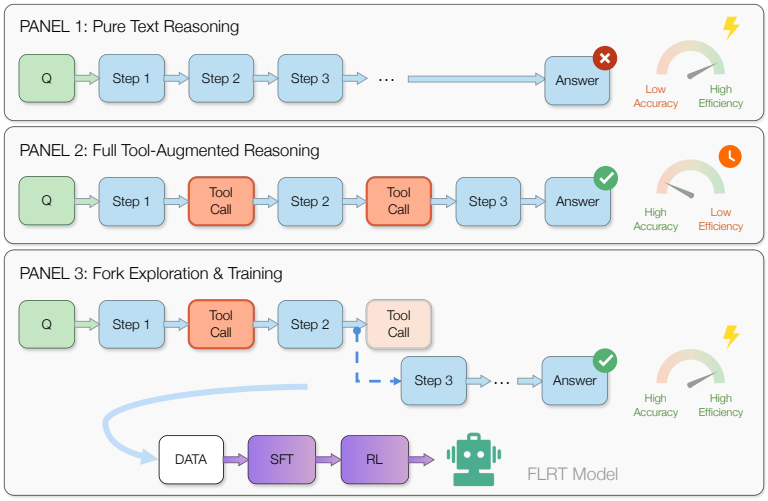


Figure 1: Motivation and overview of FLRT. A successful fork from a real intermediate state supplies a continuation without future tools. FLRT uses fork observations for supervised fine-tuning and reinforcement learning. The three panels illustrate the motivation rather than a universal accuracy or efficiency ordering.

Building on this signal, we propose **FLRT**, a two-stage training framework (Figure 1). **Fork-guided SFT** combines original prefixes with successful tool-free continuations, alongside original correct tool trajectories and naturally tool-free correct trajectories. **Fork-ranked RL** first separates correct from incorrect candidates for the same problem, then ranks correct candidates by their actual call count and measured redundancy. Deployment follows ordinary interleaved reasoning without additional Fork Exploration.

Across AIME25, AIME24, and HMMT, FLRT reaches an unweighted mean accuracy of 79.77% with 6.13 calls, compared with 76.30% and 8.90 for Base. Our contributions are:

1. **Problem and measurement.** We study within-trajectory changes in tool need, define Tool Redundancy, and observe correct continuations without future tools from real intermediate states.
2. **Learning method.** We introduce FLRT, using successful fork continuations for supervision and measured redundancy for ranking correct RL candidates.
3. **Empirical analysis.** We evaluate accuracy–tool trade-offs on three mathematical benchmarks and examine the relationship between lower call counts, redundancy, and training design.

2 Tool Redundancy in Interleaved Reasoning

Interleaved reasoning alternates internal reasoning with external tool calls and incorporates each returned result into the context. As this context changes, the need for further tools can change as well. We first examine this phenomenon, then define and measure Tool Redundancy.

2.1 Pilot Study

Given a correct tool trajectory, we select a state immediately before a call and retain the question, prior reasoning, and all earlier tool outputs. We disable all future tool access and regenerate the remaining reasoning through the final answer. We call this procedure **Fork Exploration**. Appendix Figure 5 contrasts this continuation with the original trajectory. The suffix is regenerated, rather than obtained by deleting calls while retaining the original subsequent text.

GPT-OSS-20B (OpenAI, 2025) generates one trajectory per AIME25 problem (Mathematical Association of America, n.d.); 25 of 30 answer correctly. We analyze all 25 correct trajectories. Each contains a state from which a sampled continuation answers correctly without further tools.

Observation 1. Correct interleaved reasoning trajectories can contain states from which the model completes the task correctly without further tool use, while the original trajectory continues calling tools.

2.2 Definition of Tool Redundancy

The state s_j before call j contains the original question, generated reasoning, and all preceding calls and results. It changes as the trajectory progresses.

Definition 1 (Tool Redundancy). In a correct interleaved reasoning trajectory, if the model generates a correct answer from an intermediate pre-call state with all future tools disabled, while the original trajectory continues using tools, those subsequent calls constitute Tool Redundancy in that trajectory.

This comparison is conditioned on the same trajectory and state. A high call count alone does not establish redundancy; the relevant observation is a successful continuation from the information already available. Since generation is stochastic, one failed fork does not establish that tools are necessary under other samples. A single outcome is neither a precise success-probability estimate nor an absolute property of the state.

2.3 Quantifying Tool Redundancy

We execute Fork Exploration in reverse order, starting before the last call. If the continuation is correct, we test the preceding checkpoint; we stop at the first failure. Let $r(\tau)$ denote the number of consecutive successful positions at the end of trajectory τ. For example, if forks before calls 5, 4, and 3 succeed but the fork before call 2 fails, then $r(\tau) = 3$, corresponding to calls T_3, T_4, T_5. This measures a contiguous terminal interval, not every potentially avoidable call in the trajectory; earlier successful forks may remain unvisited.

For correct trajectories $\mathcal{T}_{\text{correct}}$, with actual call count $N_{\text{tool}}(\tau)$, we report

$$R_{\text{red}} = \frac{\sum_{\tau \in \mathcal{T}_{\text{correct}}} r(\tau)}{\sum_{\tau \in \mathcal{T}_{\text{correct}}} N_{\text{tool}}(\tau)}. \tag{1}$$

We refer to this call-weighted quantity as **Tool redundancy**; suffixes distinguish alternative denominators and averaging units in Section 5. Average Tools measures the amount of external interaction, whereas R_{red} measures the fraction of calls in correct trajectories belonging to the observed contiguous redundant suffix. Fork Exploration thus supplies both successful continuations for supervision and $r(\tau)$ for trajectory ranking.

3 FLRT

We propose **Fork-Guided Learning for Reducing Tool Redundancy (FLRT)**.

3.1 Overview

FLRT uses the two outputs of Fork Exploration in two training stages (Figure 2). Fork-guided SFT replaces an original suffix with a correct continuation without further tools, preserving the complete preceding context. Fork-ranked RL evaluates candidates generated by the current policy, separates correct from incorrect answers, and ranks the correct candidates using their call count and Tool Redundancy. Fork Exploration is used for training-data construction and training-time evaluation only; the deployed model follows ordinary interleaved reasoning.

3.2 Fork-guided SFT

A successful fork yields a correct training trajectory by concatenating the full original prefix with the newly generated continuation. Earlier tool interactions remain in the prefix, while the original

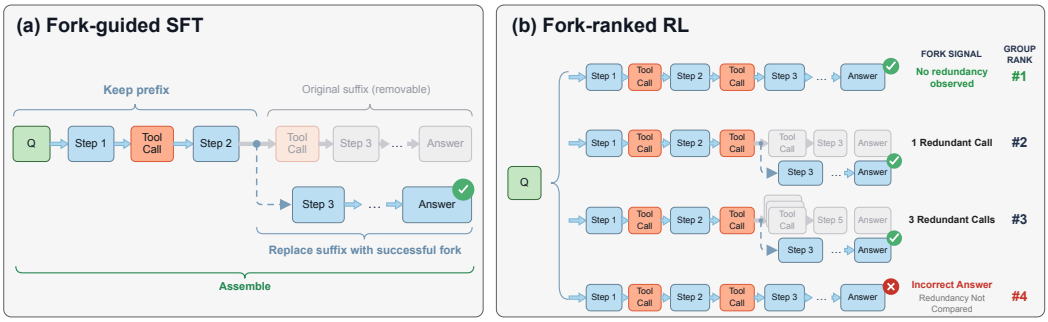


Figure 2: FLRT training. (a) A preserved prefix and successful no-future-tool continuation form a training example, mixed with original tool-correct and naturally tool-free correct trajectories. (b) Correct candidates are ranked by $N_{\text{tool}} + \alpha r$. The schematic candidates have $(N_{\text{tool}}, r) = (2, 0), (3, 1), (5, 3)$, giving scores 2, 4, and 8 when $\alpha = 1$. Incorrect candidates receive lower rewards; the displayed #4 illustrates this priority. No observed redundancy does not prove all calls necessary.

subsequent calls are replaced. The supervision set $\mathcal{D}_{\text{FLRT}}$ contains three types of correct trajectories: original tool trajectories, fork-derived trajectories, and naturally tool-free trajectories that solve the problem without tools from the start. We use the standard language-model objective

$$\mathcal{L}_{\text{SFT}} = -\sum_{(x,y)\in\mathcal{D}_{\text{FLRT}}} \sum_{t} \log \pi_\theta(y_t \mid x, y_{<t}). \tag{2}$$

No additional stopping label or prediction module is introduced. The model continues to generate reasoning, tool calls, and answers autoregressively; Fork Exploration changes the supervision trajectories.

3.3 Fork-ranked RL

For problem x, we sample a candidate group $\mathcal{G}_x = \{\tau_1, \ldots, \tau_G\}$ from the current policy and evaluate final-answer correctness. Incorrect candidates receive low rewards. For each correct candidate, the ranking score is

$$c(\tau) = N_{\text{tool}}(\tau) + \alpha r(\tau). \tag{3}$$

Let $\mathcal{C}_x$ be the correct candidates, ρ_i the ascending dense rank of $c(\tau_i)$ within $\mathcal{C}_x$, and K_x the number of distinct scores. We assign

$$R_i = \begin{cases} -1, & \tau_i \notin \mathcal{C}_x, \\ 1 - 2(\rho_i - 1)/K_x, & \tau_i \in \mathcal{C}_x. \end{cases} \tag{4}$$

Ties receive equal rewards; every correct candidate scores above -1. If none is correct, all rewards are -1. We use $\alpha = 1$ for FLRT; $\alpha = 0$ removes only the redundancy term.

We optimize these rewards with GRPO (Shao et al., 2024). For candidates sampled from π_{old}, the group-relative advantage and token probability ratio are

$$\widehat{A}_i = \frac{R_i - \text{mean}_j\, R_j}{\text{std}_j\, R_j + \delta}, \qquad u_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t} \mid h_{i,t})}{\pi_{\text{old}}(y_{i,t} \mid h_{i,t})}, \tag{5}$$

where $h_{i,t}$ includes preceding reasoning and tool returns, and $\delta > 0$ stabilizes normalization. Following PPO-style clipping (Schulman et al., 2017), the policy-gradient term is

$$\mathcal{J}_{\text{PG}}(\theta) = \mathbb{E}_{x,\mathcal{G}_x\sim\pi_{\text{old}}}\left[\left\langle \min\left(u_{i,t}\widehat{A}_i, \text{clip}(u_{i,t}, 1-\epsilon_-, 1+\epsilon_+)\widehat{A}_i\right)\right\rangle_{\mathcal{M}}\right]. \tag{6}$$

Here $\langle\cdot\rangle_{\mathcal{M}}$ averages over policy-generated tokens selected by the loss mask; tool-return tokens are context, not actions to optimize. The actor maximizes $\mathcal{J}_{\text{PG}} - \beta\mathcal{J}_{\text{KL}}$, with $\mathcal{J}_{\text{KL}}$ regularizing against the reference policy. FLRT changes the group reward, not the underlying policy-gradient principle; implementation details are given in Appendix C.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate GPT-OSS-20B-based policies (OpenAI, 2025) on AIME25 and AIME24 (Mathematical Association of America, n.d.), and HMMT (HMMT, n.d.), allowing Python tool calls during mathematical reasoning. AIME25 uses eight complete sampled trajectories per problem and also supports the subsequent redundancy analysis. We report final-answer Accuracy and Average Tools, the mean number of executed calls per complete trajectory. The three-dataset Avg is an unweighted mean across datasets, not across random seeds. Except for No-tool Prompt, methods use the interleaved tool interface. Implementation details and remaining reporting gaps are separated in Appendix C.

4.2 BASELINES

No-tool Prompt and Base. No-tool Prompt removes the tool-use prompt and tool definitions, providing an internal-reasoning control. Base is the original tool-augmented policy before efficiency training.

Reason-then-Act (RTA). We adapt the Necessary + Reason-then-Act prompt baseline of Sun et al. (2026). An instruction appended after the question asks for a brief assessment of whether the problem can be solved directly and requests tools only when necessary. Generation then follows the normal interleaved process. This is a prompt-level instruction, not a classifier or a programmatically enforced gate. Probe&Prefill, the full method from the same work, predicts tool need from pre-generation representations and steers generation through prefilled text; it is distinct from our RTA baseline.

FLRT-SFT. This variant uses only Fork-guided SFT, without subsequent RL.

GRPO. Group Relative Policy Optimization (Shao et al., 2024) constructs relative advantages from a group of same-problem samples without a separately trained value model. Our outcome-supervised baseline rewards final-answer correctness without an explicit tool-efficiency signal.

OTC-GRPO. OTC-GRPO rewards correctness and tool efficiency using complete-trajectory call counts (Wang et al., 2025a). We use its v1 formulation; FLRT additionally uses redundancy measured within trajectories. This baseline differs from our fixed per-call penalty control.

4.3 MAIN RESULTS

4.3.1 COMPARISON WITH TOOL-USE BASELINES

Method	AIME25		AIME24		HMMT		Avg	
	Acc.↑	Tools↓	Acc.↑	Tools↓	Acc.↑	Tools↓	Acc.↑	Tools↓
Training-free methods								
No-tool Prompt	59.58	0.00	70.42	0.00	48.86	0.00	59.62	0.00
Base	82.50	9.7042	81.25	6.83	65.15	10.15	76.30	8.90
Reason-then-Act	81.67	7.12	81.25	4.84	**70.08**	9.04	77.67	7.00
Non-RL training methods								
FLRT-SFT (ours)	79.17	**6.70**	79.58	5.45	66.29	7.53	75.01	6.56
RL training methods								
GRPO	81.67	7.28	80.00	5.07	64.02	7.25	75.23	6.53
OTC-GRPO	83.75	8.93	81.67	6.76	65.15	10.04	76.86	8.58
FLRT (ours)	**85.42**	6.98	**84.58**	**4.58**	69.32	**6.82**	**79.77**	**6.13**

Table 1: Main results. Accuracy is in percent; Tools counts executed calls per complete trajectory. Avg gives equal weight to the three datasets. Bold denotes the best value in each column, excluding No-tool Prompt from the minimum-call comparison. Each row is a reported evaluation operating point.

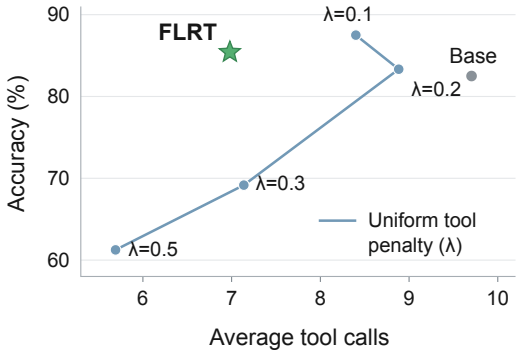


Figure 3: Accuracy–tool operating points on AIME25. Muted blue points show uniform-penalty settings; the star and gray point denote FLRT and Base. Lines connect measured settings for readability and do not represent interpolation or a continuous optimal Pareto frontier.

Tools remain useful: No-tool Prompt obtains 59.62% mean accuracy, compared with 76.30% for Base. FLRT obtains 79.77% with 6.13 calls, versus 8.90 calls for Base. Among the tool-augmented methods in Table 1, it has the highest accuracy on AIME25 and AIME24 and the fewest calls on AIME24 and HMMT. FLRT-SFT uses fewer calls on AIME25 (6.70 versus 6.98), with lower accuracy (79.17% versus 85.42%). On AIME25, its 85.42% accuracy and 6.98 calls compare with OTC-GRPO's 83.75% and 8.93 calls.

Trade-offs differ by task. On HMMT, RTA reaches 70.08%, above FLRT's 69.32%, but uses 9.04 calls rather than 6.82. FLRT does not dominate every method on both metrics for every dataset; it provides a favorable combination in the reported comparison.

4.3.2 Comparison with a Uniform Tool Penalty

A direct cost objective assigns the same penalty to every call:

$$R(\tau) = R_{\text{task}}(\tau) - \lambda N_{\text{tool}}(\tau). \tag{7}$$

This objective does not use Fork Exploration or distinguish calls by reasoning state. Figure 3 compares $\lambda \in \{0.1, 0.2, 0.3, 0.5\}$ with Base and FLRT on AIME25. The penalty sweep is non-monotonic. At $\lambda = 0.1$, accuracy is 87.50% with 8.40 calls; at 0.2, it is 83.33% with 8.88 calls. Increasing the weight to 0.3 and 0.5 reduces calls to 7.14 and 5.69, while accuracy falls to 69.17% and 61.25%. FLRT, at 85.42% and 6.98 calls, improves both metrics over the measured 0.2 and 0.3 settings. However, 0.1 is more accurate and 0.5 uses fewer calls. FLRT is therefore a nondominated point among the evaluated policies, not a universal improvement over every penalty setting. Untested penalty weights remain outside this comparison.

The uniform penalty is a competitive objective, while FLRT offers another high-accuracy, lower-call operating point. Accuracy and Average Tools alone do not identify which interactions were reduced; we next examine Tool Redundancy.

5 Analysis of Tool-use Behavior

5.1 Tool Redundancy at Multiple Aggregation Levels

We apply offline Fork Exploration to generated AIME25 trajectories. Correct trajectories are evaluated by reverse scanning, recording their contiguous redundant suffix as $r(\tau)$. We compare Base, OTC-GRPO, and FLRT.

Calls, answers, and questions. Default **Tool redundancy** is Equation 1. **Tool redundancy—All-answer** divides the same numerator by calls in all trajectories. Incorrect trajectories do not receive the same redundancy measurement, so this is the identified redundant cost as a share of total

calls, not a full redundancy assessment of incorrect answers. **Tool redundancy—Answer-mean** averages $r(\tau)/N_{\text{tool}}(\tau)$ equally over correct answers, assigning zero to tool-free correct answers. Detailed aggregation rules, including Question-mean, are given in Appendix B. In this snapshot,

Method	Acc. (%)	Tools	Tool redundancy (%) Default	All-answer	Answer-mean	Avg red. (%)†
Base	82.36	9.43	56.69	33.01	69.92	53.21
OTC-GRPO	83.75	8.93	56.99	35.95	68.10	53.68
FLRT	**85.42**	**6.98**	**45.53**	**28.58**	**57.04**	**43.72**

Table 2: Offline redundancy snapshot on AIME25. Each policy is evaluated on its own trajectories, without trajectory-level pairing. Base uses the historical pooled analysis set, not the B1 evaluation in Table 1. †Avg red. is retained from the source analysis and numerically matches the mean of the preceding three redundancy columns; it is not an independent or question-level measurement.

FLRT reduces default Tool redundancy from Base's 56.69% to 45.53%, All-answer from 33.01% to 28.58%, and Answer-mean from 69.92% to 57.04%. OTC-GRPO remains closer to Base in the default measure. Thus, lower redundancy appears in both call-weighted and answer-weighted summaries. Question-level records are still required before assessing whether these differences are distributed uniformly across problems.

5.2 Boundary Analysis: Shorter Redundant Suffixes

For a trajectory with k calls and $r(\tau) > 0$, the measured redundant suffix begins at call $k - r(\tau) + 1$, with normalized position $(k - r(\tau) + 1)/k$. This is the contiguous suffix identified by reverse scanning, not necessarily the earliest successful fork. Trajectories without an observed suffix are counted separately.

Figure 4(a) summarizes the historical boundary-analysis subset. The mean redundant suffix decreases from 4.05 to 2.72 calls, while the median normalized start shifts from 0.50 to 0.70. The fraction without an observed suffix increases from 2.89% to 6.25%. Mean total calls in this subset are 7.14 for Base and 5.98 for FLRT; it is separate from Tables 1 and 2. Median starts condition on nonempty suffixes, whereas mean lengths are separate aggregates. These results describe shorter redundant suffixes, not a causal shift in when sufficient information is acquired. Original per-trajectory records and script bindings for this historical subset remain pending.

5.3 Case Study: Verification after Obtaining a Result

Call intents. Heuristic labels derived from pre-call reasoning show that verification becomes more common after the recorded boundary: from 36.2% to 49.7% for Base and from 31.3% to 47.8% for FLRT. Repetition/recomputation does not increase consistently (9.7% to 10.1% for Base; 11.7% to 6.7% for FLRT). The shared pattern is verification of existing results, rather than a universal increase in recomputation. Intent alone does not establish redundancy.

Coin-counting example. Figure 4(c) asks how many integers $N = 1, \ldots, 1000$ can be represented with the minimum number of coins by greedy change-making using denominations $\{1, 10, 25\}$. The correct answer is 610. Base's first tool call compares greedy counts with dynamic programming and obtains this result. A fork before the call answers 600; a fork after the result answers 610 with no further tools. The original trajectory nevertheless continues checking failure intervals, patterns modulo 5 and 25, and the 390 failures, using eight calls in total. FLRT ends after obtaining the result and performing one sanity check, using two calls. These are descriptive summaries of the historical case, not verbatim recovered traces.

The failed early fork does not prove tools indispensable; the later successful fork supplies an observed path that finishes with information already obtained. FLRT thus targets avoidable subsequent interaction rather than rejecting tools from the starting state.

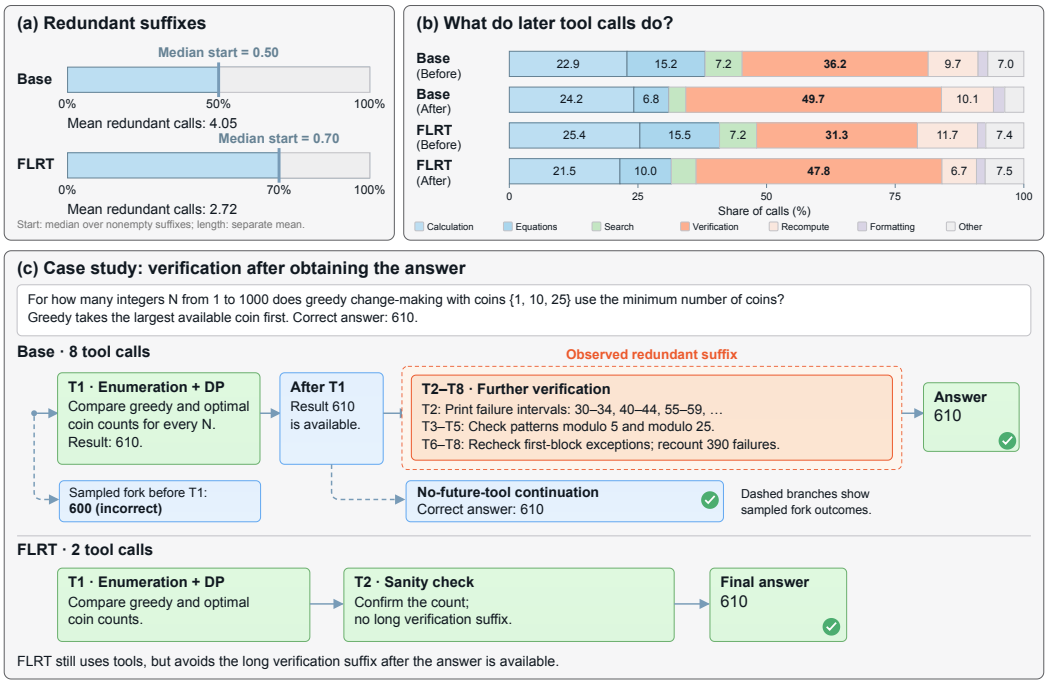


Figure 4: Tool-use analysis. (a) Median normalized starts of nonempty redundant suffixes; mean redundant call counts are annotated separately. (b) Heuristic call-intent distributions before and after the boundary recorded in the historical analysis. Each bar is normalized within its group; widths account for rounding, while labels retain reported percentages. Intent labels are descriptive, not necessity judgments. (c) A detailed coin-counting case from the historical slide summary: Base uses eight calls and FLRT two. Dashed branches represent individual fork observations, not proofs of necessity. The complete original case traces have not yet been recovered.

6 Ablations and Time Overhead

6.1 Training Stages

We compare Base, Fork-guided SFT alone, and two RL continuations after SFT on AIME25. SFT

Method	Accuracy (%)↑	Average Tools↓
Base	82.50	9.7042
Fork-guided SFT	79.17	**6.70**
Fork-guided SFT + GRPO	81.67	7.28
Fork-guided SFT + Fork-ranked RL (FLRT)	**85.42**	6.98

Table 3: Training-stage comparison on AIME25. Base uses the B1 result from Table 1; GRPO rewards correctness alone, whereas Fork-ranked RL also ranks tool use among correct candidates.

alone reduces calls from 9.7042 to 6.70 but also lowers accuracy from 82.50% to 79.17%. Continuing with GRPO reaches 81.67% and 7.28 calls; Fork-ranked RL reaches 85.42% and 6.98. Full FLRT attains the highest accuracy in this comparison, though not the fewest calls. The comparison does not include Fork-ranked RL directly from Base and is not a full factorial design; it does not establish that Fork-guided SFT is necessary for the observed gains.

6.2 Redundancy Weight

We compare $\alpha \in \{0, 1, 10\}$ in Equation 3, retaining the Fork-guided SFT initialization and the intended RL protocol. At $\alpha = 0$, the redundancy term is removed from RL ranking, but call-

count ranking and fork-derived SFT remain. This is neither standard GRPO nor removal of every fork signal. From $\alpha = 0$ to 1, mean accuracy increases from 77.70% to 79.77% at similar call

	AIME25		AIME24		HMMT		Avg		Tool redundancy
α	Acc.	Tools	Acc.	Tools	Acc.	Tools	Acc.	Tools	(%)
0	84.58	**6.49**	79.58	5.09	68.94	**6.75**	77.70	**6.11**	52.64
1	**85.42**	6.98	**84.58**	**4.58**	69.32	6.82	**79.77**	6.13	**46.96**
10	84.17	8.13	83.33	5.52	**70.83**	8.02	79.44	7.22	47.98

Table 4: Redundancy-weight ablation. Accuracy is in percent. Redundancy is measured on AIME25 only; the weight-sweep snapshot (46.96% for $\alpha = 1$) is distinct from Table 2.

counts (6.11 and 6.13), while AIME25 Tool redundancy decreases from 52.64% to 46.96%. Similar total call counts can therefore accompany different redundancy levels. Increasing α to 10 does not improve these three measures further, although HMMT accuracy is highest. The weight changes a ranking preference, not a hard call budget.

These ablations report existing operating points without multi-seed error estimates. Run-level binding of initialization checkpoints, rollout settings, training updates, and checkpoint-selection rules remains incomplete; the differences are not presented as statistically significant or strictly controlled causal component effects.

6.3 Time Overhead

We measure the additional RL update time required by online Fork Exploration. FLRT averages

Training window	GRPO (s/update)	FLRT (s/update)	Extra time
First half (updates 1–25)	650.48	730.48	+12.30%
Second half (updates 26–50)	696.35	777.93	+11.71%
Full run (updates 1–50)	673.42	754.21	+12.00%

Table 5: RL update time along the final checkpoint lineage. Both runs use eight GPUs and contain 50 unique updates. Extra time is the ratio of accumulated times over the same window minus one. The full-run row combines the two halves. Rolled-back work, restart waits, initialization outside the logged timer, prior fork-data construction, and SFT are excluded.

754.21 seconds per update versus 673.42 for GRPO, an additional 12.00%. The relative overhead is slightly lower in the second half (11.71% versus 12.30%), but the small difference does not establish a decreasing trend. Forks resume existing states. Branch counts alone do not determine overhead, and these timings do not establish a causal link between redundancy reduction and generation cost.

Deployment introduces no fork stage. FLRT uses 6.13 mean calls across the benchmarks, compared with GRPO's 6.53. End-to-end latency also depends on output length, tool execution, and scheduling; no matched inference-clock measurements are available. Fewer calls therefore do not imply an equal proportional speedup. The timings also exclude complete data-construction and SFT costs and cannot establish a training–deployment break-even point.

7 Conclusion

FLRT learns from changes in tool need within reasoning trajectories. Fork Exploration supplies correct continuations and measured redundant suffixes for SFT and RL ranking. Across three mathematical benchmarks, FLRT achieves 79.77% mean accuracy with 6.13 calls, versus 76.30% and 8.90 for Base. The analyses distinguish total call counts from observed redundancy. Deployment requires no Fork Exploration.

AI USE STATEMENT

We used generative AI tools to assist with method development, experimental code, data-related work, and discussions of result interpretation. We also used these tools for literature search, bibliography preparation, manuscript translation and editing, figure design and vector-rendering code, and LaTeX typesetting. AI-generated images served as visual design references rather than sources of experimental measurements. The authors take responsibility for the final content of this work, including its methods, data, code, results, claims, and references.

REFERENCES

Liuji Chen, Dianxing Tang, Xing Shi, Dingshuo Chen, Qiang Liu, Shu Wu, and Liang Wang. Learning When Not to Act: Mitigating Tool Abuse in Agentic Reinforcement Learning, 2026. URL https://arxiv.org/abs/2606.02132v2. Version 2606.02132v2; preprint edition.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, 2025. URL https://arxiv.org/abs/2501.12948v1. Version 2501.12948v1; preprint edition.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided Language Models, 2022. URL https://arxiv.org/abs/2211.10435v1.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving, 2023. URL https://arxiv.org/abs/2309.17452v4. Version 2309.17452v4; preprint edition.

HMMT. Harvard–MIT Mathematics Tournament, n.d. URL https://www.hmmt.org/. Official competition website. Accessed September 26, 2026.

Mathematical Association of America. MAA Invitational Competitions: American Invitational Mathematics Examination (AIME), n.d. URL https://maa.org/

maa-invitational-competitions/. Official competition website. Accessed September 26, 2026.

OpenAI. gpt-oss-120b & gpt-oss-20b Model Card. Technical report, OpenAI, August 2025. URL https://cdn.openai.com/pdf/419b6906-9da6-406c-a19d-1bb078ac7637/oai_gpt-oss_model_card.pdf.

Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi Chen, Avirup Sil, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. SMART: Self-Aware Agent for Tool Overuse Mitigation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4604–4621, 2025. doi: 10.18653/v1/2025.findings-acl.239. URL https://doi.org/10.18653/v1/2025.findings-acl.239.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, 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*, pp. 68539–68551, 2023. doi: 10.52202/075280-2997. URL https://doi.org/10.52202/075280-2997.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017. URL https://arxiv.org/abs/1707.06347v2.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024. URL https://arxiv.org/abs/2402.03300v3. Version 2402.03300v3; preprint edition.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A Flexible and Efficient RLHF Framework, 2024. URL https://arxiv.org/abs/2409.19256v2.

Chung-En Sun, Linbo Liu, Ge Yan, Zimo Wang, and Tsui-Wei Weng. LLM Agents Already Know When to Call Tools – Even Without Reasoning, 2026. URL https://arxiv.org/abs/2605.09252v2. Version 2605.09252v2; preprint edition.

Leitian Tao, Baolin Peng, Wenlin Yao, Tao Ge, Hao Cheng, Mike Hang Wang, Jianfeng Gao, and Sharon Li. TRACE: Turn-level Reward Assignment via Credit Estimation for Long-Horizon Agents, 2026. URL https://arxiv.org/abs/2607.13988v1. Version 2607.13988v1; preprint edition.

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. OTC: Optimal Tool Calls via Reinforcement Learning, 2025a. URL https://arxiv.org/abs/2504.14870v1. Version 2504.14870v1; preprint edition.

Hongru Wang, Boyang Xue, Baohang Zhou, Tianhua Zhang, Cunxiang Wang, Huimin Wang, Guanhua Chen, and Kam-Fai Wong. Self-DC: When to Reason and When to Act? Self Divide-and-Conquer for Compositional Unknown Questions. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6510–6525. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.naacl-long.331. URL https://aclanthology.org/2025.naacl-long.331/.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-Of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35*, pp. 24824–24837, 2022. doi: 10.52202/068431-1800. URL https://doi.org/10.52202/068431-1800.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models, 2022. URL https://arxiv.org/abs/2210.03629v3. Version 2210.03629v3; preprint edition.

Haiyue Zhang. Credit Without Ground Truth: Auditing Step-Level Credit Assignment in LLM Agents Against Executed Replay, 2026. URL https://arxiv.org/abs/2608.19760v2. Version 2608.19760v2; preprint edition.

A Related Work

A.1 Tool-augmented Mathematical Reasoning

External tools supply computation, retrieval, and execution beyond a model's parameters. PAL generates intermediate programs and delegates their execution to an interpreter (Gao et al., 2022). ReAct interleaves reasoning, actions, and observations so that newly obtained information informs subsequent reasoning (Yao et al., 2022). For mathematics, ToRA combines natural-language reasoning with program execution and trains on tool-interaction trajectories. Its Output Space Shaping also preserves prefixes of incorrect trajectories and uses a teacher to regenerate later steps into correct training examples (Gou et al., 2023). These methods establish foundations for interleaved tool reasoning and continuation-based training. FLRT focuses on subsequent interaction in correct trajectories: after reasoning and tool results have entered the context, can the model finish if future tools are disabled?

A.2 Tool-efficiency Optimization

OTC-GRPO multiplies task reward by a tool-efficiency coefficient. In the v1 formulation used here, it takes the minimum call count among correct same-problem candidates as a local approximation to the target count and constructs a nonlinear reward from its relationship to the current count (Wang et al., 2025a). It therefore differs from a fixed linear per-call penalty: reward also depends on the candidate reference and task outcome.

EAPO includes trajectories generated without tools from the beginning, compares them with tool-augmented solutions for the same problem, and adjusts efficiency rewards with problem difficulty (Chen et al., 2026). These approaches already use tool availability and interaction cost to improve policies. FLRT instead measures continuations from states that retain information accumulated during a real solution; beyond total calls, Fork-ranked RL uses the contiguous terminal redundancy identified by reverse scanning.

A.3 Dynamic Tool-need Decisions

Tool decisions can apply to an entire question or recur during problem solving. SMART decomposes composite questions into subgoals when constructing supervision, labels whether each step relies on parametric knowledge or tools, and includes rationales for using or avoiding tools. Later subgoals can use previous reasoning and tool results (Qian et al., 2025). Self-DC uses confidence on a question or subquestion to choose internal generation, external retrieval, or further decomposition, recursively solving and combining subanswers (Wang et al., 2025b). Neither approach is limited to a single decision at the problem's starting state.

When2Tool constructs tool-need labels from success or failure under forced no-tool answering. Probe&Prefill predicts these labels from pre-generation hidden states and steers generation with prefilled text (Sun et al., 2026). Its default soft prefill can be overridden by the model, while hard prefill enforces an output format; the benchmark also includes dependency chains requiring hop-level decisions. Fork Exploration instead executes a remaining-solution test from a real intermediate state, preserving prior context and disabling every future tool. Successful continuations train Fork-guided SFT, while deployment uses the ordinary tool-augmented policy.

A.4 Trajectory Interventions and Contribution Assessment

Toolformer filters training data by comparing subsequent token-prediction losses with the API result, with the call alone, and without the call, retaining calls whose returned information sufficiently improves the loss (Schick et al., 2023). TRACE scores prefixes using a frozen reference model's conditional probability of the correct answer and assigns turn-level rewards through value differences between successive tool-boundary states (Tao et al., 2026). These signals concern predictive utility or state scores, rather than actually generating a complete correct continuation with future tools disabled.

Executed Replay repeatedly executes an original action from a replayable ALFWorld decision state and compares terminal outcomes with those from policy-supported alternative actions to audit step-

level credit signals (Zhang, 2026). This changes the current action to examine its relationship with subsequent outcomes. Fork Exploration preserves preceding actions and returns, intervenes on all future tool access, and evaluates the generated final answer. FLRT uses observed successful no-future-tool paths and their contiguous terminal measurement, rather than attributing contribution to each past action. A failed fork does not establish absolute necessity.

B Metrics and Aggregation

For a complete trajectory set $\mathcal{T}_D$, final-answer evaluator E, and $M_D = |\mathcal{T}_D|$,

$$\mathrm{Acc}(D) = \frac{100}{M_D} \sum_{\tau \in \mathcal{T}_D} E(\tau), \qquad \mathrm{Tools}(D) = \frac{1}{M_D} \sum_{\tau \in \mathcal{T}_D} N_{\mathrm{tool}}(\tau). \tag{8}$$

With equal samples per question, accuracy is the average per-question success fraction, not pass@k. Dataset-level Avg gives equal weight to each of the three benchmarks. For positive denominators,

$$R_{\mathrm{all}} = \frac{\sum_{\tau \in \mathcal{T}_{\mathrm{correct}}} r(\tau)}{\sum_{\tau \in \mathcal{T}} N_{\mathrm{tool}}(\tau)}, \tag{9}$$

$$R_{\mathrm{answer}} = \frac{1}{|\mathcal{T}_{\mathrm{correct}}|} \sum_{\tau \in \mathcal{T}_{\mathrm{correct}}} h(\tau), \qquad h(\tau) = \begin{cases} r(\tau)/N_{\mathrm{tool}}(\tau), & N_{\mathrm{tool}}(\tau) > 0, \\ 0, & N_{\mathrm{tool}}(\tau) = 0. \end{cases} \tag{10}$$

These correspond to All-answer and Answer-mean. All-answer retains the numerator measured on correct trajectories; it does not add an unmeasured incorrect-trajectory redundancy term. Default Tool redundancy weights correct trajectories by their call counts, while Answer-mean weights correct answers equally. Question-mean is defined below. If an entire pooled denominator is zero, a pooled ratio is undefined; the stated zero conventions apply to individual answer and question terms.

Tool redundancy—Question-mean first pools correct trajectories per question and then averages equally over questions with at least one correct trajectory:

$$R_{\mathrm{question}} = \frac{1}{|\mathcal{Q}_{\mathrm{correct}}|} \sum_{x \in \mathcal{Q}_{\mathrm{correct}}} \frac{\sum_{\tau \in \mathcal{T}_{x,\mathrm{correct}}} r(\tau)}{\sum_{\tau \in \mathcal{T}_{x,\mathrm{correct}}} N_{\mathrm{tool}}(\tau)}. \tag{11}$$

A question whose correct trajectories are all tool-free contributes zero; questions without a correct trajectory are excluded. These summaries distinguish call-weighted cost from the experience of an equally weighted correct answer or question.

C Fork Protocol and Reporting Scope

C.1 Fork and Reverse Scan

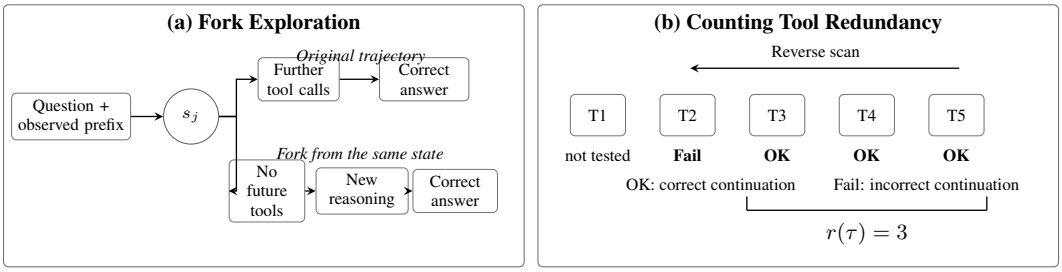


Figure 5: Fork Exploration holds the observed prefix fixed while generating a new continuation with future tools disabled (a). Reverse scanning (b) stops at the first failure; successful forks before T_5, T_4, T_3 yield $r(\tau) = 3$.

Restore the original pre-call checkpoint, including the question, preceding reasoning, and earlier tool outputs. The current call's not-yet-produced return and the original future suffix are excluded.

Disable future tools, generate a continuation, and evaluate its final answer against the reference. The reference is used by the evaluator, not supplied to generation.

For k calls, initialize $r = 0$ and visit checkpoints $k, k-1, \ldots, 1$. Each fork starts independently from its original prefix. Increment r after a correct continuation and stop at the first failure. If every fork succeeds, $r = k$. The number of generated branches is k when all succeed and $r + 1$ otherwise; this count alone does not measure training time.

C.2 Reward Interface

Evaluate every candidate's final answer. For correct candidates, compute $N_{\text{tool}} + \alpha r$, dense-rank distinct scores in ascending order, and map lower scores to higher rewards. Incorrect candidates receive rewards below those of correct candidates. Equation 4 gives the numerical mapping. The local VERL implementation (Sheng et al., 2024) normalizes rewards within prompt groups and supports token-masked loss aggregation, asymmetric clipping, and a negative-advantage dual-clip safeguard in addition to Equation 6. The supplied RL recipe enables reference-policy KL loss.

D Limitations

Fork outcomes depend on the policy, decoding, context, tool environment, and evaluator. Single samples and reverse scanning provide bounded observations, not exact success probabilities or exhaustive identification of avoidable calls. The reported operating points lack multi-seed uncertainty estimates, and the ablations do not isolate every causal component. The discrete penalty sweep cannot establish dominance over an entire family of cost objectives.

The study concerns mathematical reasoning with Python. Final-answer correctness does not verify every intermediate step, and the findings are not established for retrieval or other environments. Additional training computation is only partially timed; tool counts are not a complete proxy for deployment latency or total train–deploy cost. Historical analysis subsets and unresolved source bindings are explicitly identified above.