Learning from the Irrecoverable: Error-Localized Policy Optimization for Tool-Integrated LLM Reasoning

Anonymous ACL submission

Abstract

Tool-integrated reasoning (TIR) enables LLM agents to solve tasks through planning, tool use, and iterative revision, but outcome-only reinforcement learning in this setting suffers from sparse, delayed rewards and weak step-level credit assignment. In long-horizon TIR trajectories, an early irrecoverable mistake can determine success or failure, making it crucial to localize the *first irrecoverable step* and leverage it for fine-grained credit assignment. We propose **Error-Localized Policy Optimization (ELPO)**, which localizes the first irrecoverable step via binary-search rollout trees under a fixed rollout budget, converts the resulting tree into stable learning signals through hierarchical advantage attribution, and applies error-localized adaptive clipping to strengthen corrective updates on the critical step and its suffix. Across TIR benchmarks in math, science QA, and code execution, ELPO consistently outperforms strong Agentic RL baselines under comparable sampling budgets, with additional gains in Pass@K and Major@K scaling, rollout ranking quality, and tool-call efficiency. Our code is publicly released for reproducibility at `https://anonymous.4open.science/r/ELPO-7C19`.

1 Introduction

Reinforcement Learning with Verifiable Rewards (RLVR) substantially improves LLM performance on complex reasoning tasks by biasing generation toward correct solution trajectories (Schulman et al., 2017; Rafailov et al., 2023; Shao et al., 2024; Wen et al., 2025; Zhang et al., 2025c). Yet evidence suggests RLVR mainly reweights behaviors already present in the base model and may not reliably expand its reasoning boundary (Yue et al., 2025). This motivates agentic reinforcement learning (Agentic RL), where LLMs solve tasks by planning, invoking external tools, and iteratively revising intermediate decisions rather than relying solely on internal chain-of-thought generation (Zhang et al., 2025a).

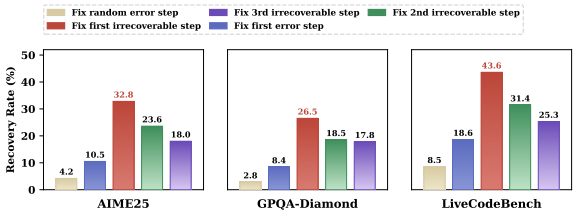

Figure 1: For each initially failed trajectory (Pass@16= 0), we edit exactly one step (random error, first error, or the 1st/2nd/3rd irrecoverable step) and resample completions. Recovery is counted when the repaired run reaches Pass@16= 1.

Tool-integrated reasoning (TIR), though enhances LLM's capability, brings new challenges to current outcome reward based reinforcement techniques, represented by GRPO (Shao et al., 2024). These outcome-based methods introduce sparsity and delay in reward signals. The problems are less pronounced in general reasoning tasks but are significantly amplified in Agentic RL (Zhang et al., 2025b). First, agentic systems rely on open-ended autonomous exploration, which dramatically enlarges the action and tool-use space. Sparse terminal rewards therefore exacerbate the exploration bottleneck. Second, agentic decision-making typically involves long, multi-step and hierarchical planning, where delayed outcome rewards fail to provide effective credit assignment across different stages and steps. More critically, relying solely on outcome supervision can trap the agent in a repeated "trial–error–no-feedback" loop, making it difficult to ensure stable and directed learning. In summary, we argue that designing finer-grained process reward signals is crucial for Agentic RL.

Most of the recent works on Agentic RL spend their effort on outcome-level reward design (Jin et al., 2025; Li et al., 2025; Qian et al., 2025; Wang et al., 2025b; Singh et al., 2025; Huang et al., 2025; Wei et al., 2025; Lin and Xu, 2025; Dong et al., 2025; Lu et al., 2025), while only a few address the sparsity of process reward signals (Feng et al.,

2025; Ji et al., 2025; Wang et al., 2025a). Overall, enabling fine-grained process supervision and reliable credit assignment in long-horizon TIR remains under-explored, which in turn constrains further progress in Agentic RL.

In this work, we directly confront the core of the problem and propose **Error-Localized Policy Optimization (ELPO)**, a novel approach that automatically identifies critical error steps during rollout and leverages them to provide finer-grained signals. The motivation stems from a key experimental finding (shown in Figure 1): localizing and correcting the **first irrecoverable error step** significantly increases the model's probability of producing the correct answer, whereas fixing a random error provides much less benefit. Thus, this motivates us to treat the recoverability of erroneous steps as a highly discriminative fine-grained signal.

However, incorporating this insight into Agentic RL training is non-trivial and requires addressing two key challenges. First, *Budget bottleneck*: efficiently identifying irrecoverable error steps without exhaustive rollouts. Second, *Process signal generation*: deriving stable and discriminative supervision from localized errors. To this end, we propose: (i) binary-tree-based Error Localization (BEL) (section 4.1) with entropy-gap-guided path pruning for efficient error localization; (ii) a hybrid trajectory- and branch-level credit assignment scheme (section 4.2) and (iii) error-localized adaptive clipping (section 4.3) to strengthen corrective updates on the localized critical step. With these techniques, our method consistently outperforms strong Agentic RL baselines across diverse TIR benchmarks.

In summary, the key contributions are as follows:

- We identify an **error-localization gap** in outcome-only long-horizon TIR: sparse terminal rewards fail to isolate the **first irrecoverable step**, yielding diffuse credit assignment and weak corrections.
- We propose **Error-Localized Policy Optimization (ELPO)**, which localizes the first irrecoverable step via a binary-search rollout procedure under a fixed rollout budget, and turns this signal into error-localized advantages and targeted policy updates.
- Across diverse TIR benchmarks, ELPO consistently outperforms strong Agentic RL baselines. Beyond final accuracy, analyses on Pass@k and Major@k, rollout ranking quality, and tool-call efficiency further validate its effectiveness and stability.

2 Related Work

2.1 Tool-integrated Reasoning Agents

Recent studies have explored enabling LLMs to act as decision-making agents equipped with external tools such as search engines, calculators, code interpreters, and APIs (Yao et al., 2022; Schick et al., 2023). Early approaches primarily relied on prompting strategies such as ReAct (Yao et al., 2022) and Reflexion (Shinn et al., 2023) to interleave reasoning with tool calls, while later work examined finetuning LLMs on tool-usage traces to improve robustness in multi-step reasoning scenarios (Qin et al., 2023; Tang et al., 2023). This line of work highlights both the promise and the difficulty of training reliable tool-augmented agents, particularly when errors made early in the reasoning process propagate through subsequent steps.

2.2 Reinforcement Learning for LLM Agents

Reinforcement learning has emerged as a key technique for improving long-horizon reasoning in LLMs. Group-based policy optimization methods such as GRPO (Shao et al., 2024) and DAPO (Yu et al., 2025a) avoid explicit value networks by normalizing rewards among multiple rollouts for the same prompt, enabling large-scale RL training for reasoning, search, and tool-use tasks. Building on this foundation, early Agentic RL systems such as ToRL (Li et al., 2025) and Search-R1 (Jin et al., 2025) apply outcome-based RL to tool-augmented reasoning, showing that verifiable terminal supervision can elicit effective tool use (e.g., code execution and web search). Subsequent work (Huang et al., 2025; Feng et al., 2025; Lin and Xu, 2025) improves learning efficiency and stability by introducing finer-grained reward design or advantage shaping on top of outcome supervision.

Tree-structured rollouts have also become an active direction for extracting richer training signals. Tree-GRPO (Ji et al., 2025) constructs a rollout tree via stochastic expansion and leverages the tree structure for fine-grained advantage estimation, but its random branching can be sensitive to the initial trajectory quality. More recent methods (Lu et al., 2025; Dong et al., 2025) further use entropy-guided expansion to steer branching and encourage more diverse sampling under the same rollout budget.

Despite this progress, existing approaches largely remain outcome-driven and do not explicitly localize the *first irrecoverable step* in long, interdependent tool-use trajectories. As a result,

even with step-aware heuristics or tree-based advantages, credit can still be spread across many non-causal steps, and the agent receives no direct signal about *where* the trajectory first becomes irrecoverable under a fixed sampling budget, which limits how targeted and stable policy updates can be for long-horizon TIR.

3 Preliminaries

3.1 Problem setup

We consider a tool-augmented reasoning setting where an LLM acts as an agent to solve multi-step tasks given a task description $x \in \mathcal{X}$. At each step $t = 1, \ldots, T$, the agent observes a state $\boldsymbol{s}_t \in \mathcal{S}$ and produces a textual action $\boldsymbol{a}_t \in V^n$, where V is the token vocabulary and n is the maximum step length. The environment executes the requested tool calls (e.g., code interpreter, search engine), returns the corresponding outputs, and deterministically updates the state to $\boldsymbol{s}_{t+1}$. A full episode induces a trajectory $\tau = \{(\boldsymbol{s}_1, \boldsymbol{a}_1), \ldots, (\boldsymbol{s}_T, \boldsymbol{a}_T)\}$, governed by an LLM policy $\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t, x)$. Rewards are only provided at the end of the episode: a terminal scalar reward $R(\tau) \in \mathbb{R}$ is assigned based on the correctness of the final answer.

3.2 Group-based Agentic RL

Recent RL works converge on a group-based training recipe. For each task input x and tool set $\mathcal{T}$, the old policy $\pi_{\theta_{\text{old}}}$ samples a group of trajectories $\{o_i\}_{i=1}^G$, each with an outcome-only terminal reward R_i. The advantage of trajectory o_i is computed by normalizing rewards within the group using $\hat{A}_i = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^G)}{F_{\text{norm}}(\{R_j\}_{j=1}^G)}$.

The GRPO-style objective is

$$\max_\theta \ \mathbb{E}_{x,\{o_i\}}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{\sum_t m_{i,t}}\sum_t m_{i,t}\, L_{\text{CLIP}}(\theta, i, t)\right], \tag{1}$$

where $m_{i,t} \in \{0, 1\}$ masks out environment feedback tokens so that the loss only accumulates over agent-generated tokens. The clipped surrogate loss follows PPO-style:

$$L_{\text{CLIP}}(\theta, i, t) = \min\Big(\rho_{i,t}(\theta)\hat{A}_i, \ \text{clip}\big(\rho_{i,t}(\theta), 1-\varepsilon_{low}, 1+\varepsilon_{high}\big)\hat{A}_i\Big) \tag{2}$$

with token-level importance ratio $\rho_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t}|o_{i,<t}, x; \mathcal{T})}{\pi_{\theta_{\text{old}}}(o_{i,t}|o_{i,<t}, x; \mathcal{T})}$.

4 Methodology

To address sparse terminal rewards in TIR, which assign the same outcome signal to all steps and leave the failure-causing decision unidentified, we propose Error-Localized Policy Optimization (ELPO). ELPO builds a binary-search rollout tree to localize the first irrecoverable step with only $O(\log K)$ anchor probes, enabling targeted advantage attribution and updates. As shown in Figure 2, ELPO consists of three core designs:

(1) **Tree Rollout Based on Binary Error Localization**: ELPO localizes the first irrecoverable step via tree rollouts with binary-search budget control.

(2) **Fine-Grained Advantage Attribution**: ELPO uses the rollout tree to assign per-step advantages from local branch comparisons and global trajectory ranking, reducing diffuse credit under outcome-only rewards.

(3) **Error-Localized Adaptive Clipping (ELC)**: Using the localized error step on the rollout tree, ELPO relaxes the clipping constraint for the error step and its generated suffix, allowing stronger corrective updates while preserving policy updating stability.

4.1 Tree Rollout Based on Binary Error Localization (BEL)

In ELPO, we use a binary-search rollout tree to pinpoint the first irrecoverable step in a failed trajectory. Concretely, given a failed trajectory, we repeatedly select the middle step as an **anchor step**. The anchor step and all preceding steps are fixed as a prefix, and several completions are sampled for the remaining suffix. If any suffix completion succeeds, the prefix is recoverable and we search the later half; otherwise it is irrecoverable and we search the earlier half. We repeat this process until the range collapses to a single step, which localizes the first irrecoverable step with only $O(\log K)$ anchor probes rather than exhaustively testing every step. The main practical challenge is budget control: binary localization requires extra suffix completions, but the per-input rollout budget must remain fixed. We address this with two complementary mechanisms.

Trajectory Selection by Entropy Gap. Under a fixed budget, we run localization on only one informative failed trajectory per cycle because many

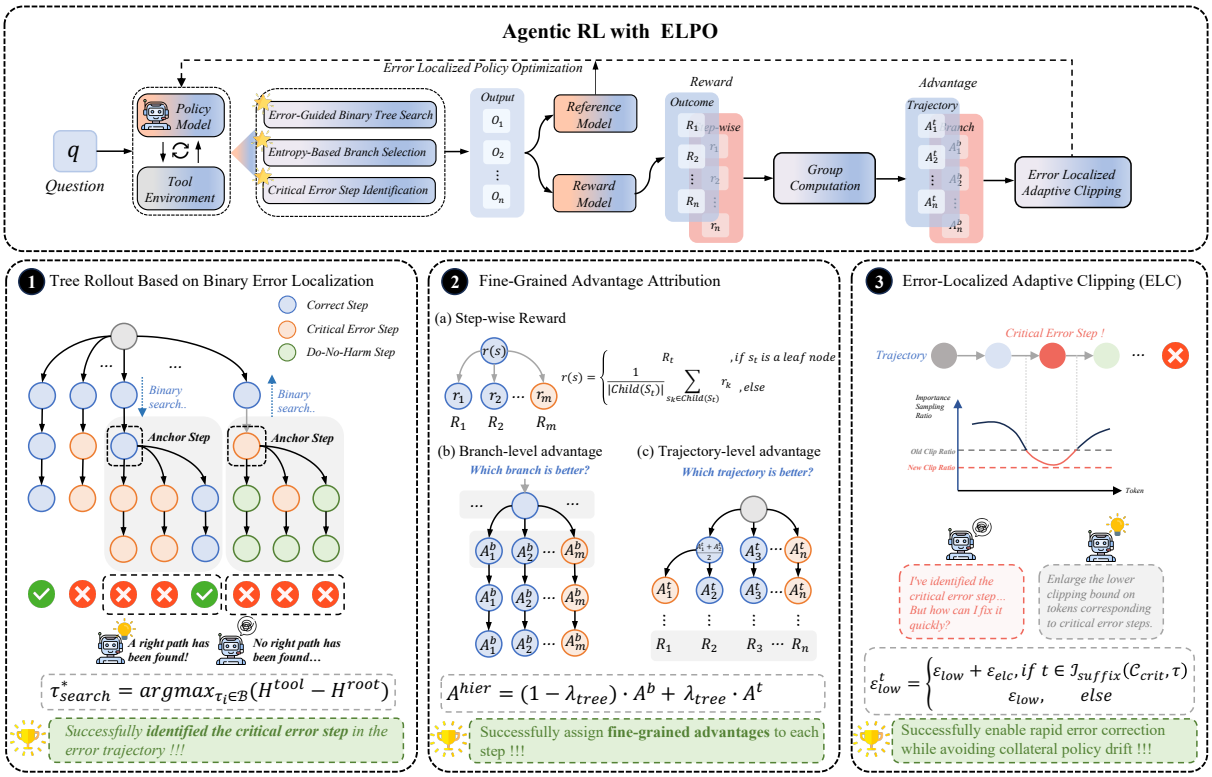

Figure 2: The overview of Error-Localized Policy Optimization (ELPO).

failures are redundant and provide little additional contrast for learning. We maintain a buffer $\mathcal{B}$ that stores the most recent erroneous trajectories from the current sampling. Guided by recent study (Lu et al., 2025)'s observation that token entropy typically spikes after tool-call feedback, we use the entropy gap between the tokens generated right after the task prompt and those generated right after tool-call feedback to decide which error trajectories deserve further branching. For each trajectory $\tau \in \mathcal{B}$, we compute token entropy of the i th token as

$$H_i = -\sum_{j=1}^{V} p_{i,j} \log p_{i,j}, \ \mathbf{p}_i = \pi_\theta(\cdot \mid y_{<i}, x; \mathcal{T}), \tag{3}$$

and then compute H^{root} as the average token entropy of the model's initial response to the question, and H^{tool} as the average entropy over all tool calls. ELPO then selects a single trajectory

$$\tau^*_{\text{search}} = \arg\max_{\tau_i \in \mathcal{B}} \left(H^{\text{tool}}(\tau_i) - H^{\text{root}}(\tau_i)\right), \tag{4}$$

on which we subsequently run binary search. Once the selected error trajectory τ^*_{search} is used for branching, we clear the buffer and start accumulating new error samples for the next cycle.

Adaptive Suffix Sampling at Anchor Steps. Anchor steps with larger post-tool uncertainty are more likely to be near the critical error. We therefore allocate more suffix completions to anchors that are more likely to lie near the critical error, and probe low-uncertainty anchors with fewer samples. Specifically, the number of suffix completions at the m-th anchor is set to

$$X_m = X_{\min} + (X_{\max} - X_{\min}) \cdot \sigma\big(\beta(H_m^{\text{tool}} - H^{\text{root}})\big), \tag{5}$$

where H^{root} is the entropy of the initial response and H_m^{tool} the step-level entropy after the m-th tool call. Large positive entropy gaps trigger more local sampling, while stable, low-uncertainty steps are probed minimally.

Through this binary-search mechanism, ELPO identifies the first irrecoverable step in only $O(\log K)$ anchor evaluations, dramatically reducing rollout cost compared to naively checking all K steps. The pseudo-code is provided in Algorithm 1 in Appendix A. An example of BEL rollout tree is provided in Appendix B.

4.2 Fine-Grained Advantage Attribution

BEL identifies *where* a trajectory first becomes irrecoverable, but effective policy optimization further requires assigning credit *how much* each deci-

sion contributes to success or failure. Therefore, we leverage the binary search rollout tree to perform fine-grained advantage attribution. We compute advantages from two sources: **branch-level** sibling comparisons and **trajectory-level** rollout ranking, and combine them into a hierarchical per-step advantage. This two-granularity attribution yields localized yet globally consistent learning signals under outcome-only rewards.

Branch-Level Advantage Attribution. Branch-level advantage is used to compare the relative quality of local branches under the same prefix. To support branch comparisons, a **step-wise reward** is defined for each node in the rollout tree $\mathcal{G}_x = (\mathcal{S}, \mathcal{E})$, estimating the expected terminal outcome of its prefix under the sampled continuations. Let $R(s)$ denote the terminal reward when s is a leaf node. The step-wise reward $r(s)$ is defined recursively:

$$r(s) = \begin{cases} R(s), & \text{if } s \text{ is a leaf node}, \\ \dfrac{1}{|\mathrm{Child}(s)|} \sum\limits_{c \in \mathrm{Child}(s)} r(c), & \text{otherwise}, \end{cases} \tag{6}$$

where $\mathrm{Child}(s)$ denotes the direct children of s in $\mathcal{G}_x$.

For any branching node s with $\mathrm{Child}(s) = \{c_1, \ldots, c_M\}$ and $M > 1$, the branch-level advantage of each child branch c_i is computed by normalizing $\{r(c)\}_{c \in \mathrm{Child}(s)}$:

$$A^b_{c_i} = \frac{r(c_i) - \mu_b}{\sigma_b + \epsilon}, \tag{7}$$

where μ_b and σ_b are the mean and standard deviation over $\{r(c)\}_{c \in \mathrm{Child}(s)}$. This branch-level normalization provides localized credit signals exactly at decision points where alternative actions diverge.

Trajectory-Level Advantage Attribution. To provide a global preference signal by ranking complete rollouts for the same input, ELPO additionally defines a trajectory-level advantage over full trajectories. For a given input x, the binary-search rollout tree $\mathcal{G}_x$ contains a set of N completed leaf trajectories $\{\tau^{(i)}\}_{i=1}^N$, each associated with a terminal reward $\{R^{(i)}\}_{i=1}^N$. Following group-based RL, we normalize these rewards within the same input and define

$$A^t(\tau^{(i)}) = \frac{R^{(i)} - \mu_t}{\sigma_t + \epsilon}, \tag{8}$$

where μ_t and σ_t are the mean and standard deviation of $\{R^{(i)}\}_{i=1}^N$.

In the rollout tree, an internal node s may be shared by multiple trajectories. We assign node-level trajectory credit by averaging the trajectory-level advantages passing through s:

$$A^t(s) = \frac{1}{|\mathcal{T}(s)|} \sum_{i \in \mathcal{T}(s)} A^t(\tau^{(i)}), \tag{9}$$

where $\mathcal{T}(s)$ denotes the set of trajectories whose paths include s. This term provides a coarse but stable signal on long linear segments, complementing the localized supervision from branch-level advantages.

Hierarchical Advantage Attribution. Branch-level advantage localizes credit at branching decisions, while the trajectory-level term provides a stable global signal. We combine them by a weighted sum:

$$A^{hier}(s) = \lambda_{\text{tree}}\, A^b(s) + (1 - \lambda_{\text{tree}})\, A^t(s), \tag{10}$$

where $\lambda_{\text{tree}} \in [0, 1]$ controls the trade-off. When $\lambda_{\text{tree}} = 0$, ELPO reduces to trajectory-level group-based RL; when $\lambda_{\text{tree}} = 1$, learning is driven purely by local branch comparisons. We use A^{hier} in the standard GRPO-style surrogate objective in place of the original group-level advantage.

4.3 Error-Localized Adaptive Clipping (ELC)

Standard GRPO applies a uniform clipping ratio to all agent-generated tokens, which can limit the policy update strength even after BEL has identified first irrecoverable steps on the rollout tree. Error-Localized Adaptive Clipping (ELC) relaxes the lower clipping bound for the critical step and its generated suffix. This allows larger corrective updates on the error-related suffix while preserving GRPO's clipping-based stability.

Concretely, we keep the PPO-style clipping form but replace the constant ratio lower bound $1 - \varepsilon_{low}$ in Eq. (2) with a token-dependent bound $1 - \varepsilon^t_{\text{low}}$, where $\varepsilon^t_{\text{low}}$ is increased on the localized critical-step suffix:

$$\varepsilon^t_{\text{low}} = \begin{cases} \varepsilon_{low} + \varepsilon_{\text{elc}}, & t \in \mathcal{I}_{\text{suffix}}(t_{\text{crit}}, \tau), \\ \varepsilon_{low}, & \text{else}, \end{cases} \tag{11}$$

where $\mathcal{C}_{\text{crit}}$ is the set of first irrecoverable steps returned by Algorithm 1, and $t_{\text{crit}} \in \mathcal{C}_{\text{crit}}$ denotes the localized critical step on trajectory τ. $\mathcal{I}_{\text{suffix}}(t_{\text{crit}}, \tau)$ denotes the set of agent-generated token positions in τ from step t_{crit} to the end of the trajectory, namely all tokens in $\{\boldsymbol{a}_t\}_{t=t_{\text{crit}}}^T$. To see the effect,

consider the clipped term $\min(\rho A,\ \mathrm{clip}(\rho, 1-\varepsilon_{\text{low}}^{t}, 1+\varepsilon_{\text{high}})A)$. When $A < 0$, decreasing $1-\varepsilon_{\text{low}}^{t}$ allows smaller ratios $\rho < 1$ to remain unclipped, which increases the magnitude of the negative update and penalizes error-inducing suffix tokens more aggressively. For $A > 0$, we keep the upper bound $1+\varepsilon_{\text{high}}$ unchanged, so the effect is limited mainly to discouraging the error suffix rather than amplifying positive updates. The hyper-parameter $\varepsilon_{\text{elc}} \geq 0$ controls the relaxation degree, enabling stronger corrections on the error-related suffix while preserving the training stability.

4.4 Training objective of ELPO

ELPO follows the GRPO training objective in Eq. (1), but modifies the token-level surrogate in two places: we use the hierarchical advantage A^{hier} in place of the group-level advantage $\hat{A}_i$, and we replace the constant lower clipping bound $1-\varepsilon_{low}$ with the token-dependent bound $1-\varepsilon_{\text{low}}^{t}$ in Eq. (11). This yields the ELPO clipped surrogate loss

$$L_{\text{CLIP}}^{\text{ELPO}}(\theta, i, t) = \min\Big(\rho_{i,t}(\theta)A^{\text{hier}},\ \mathrm{clip}\big(\rho_{i,t}(\theta), 1-\varepsilon_{\text{low}}^{t}, 1+\varepsilon_{high}\big)A^{\text{hier}}\Big) \tag{12}$$

which is plugged into Eq. (1) with the same masking scheme.

5 Experimental Setup

5.1 Datasets

We evaluate ELPO across three long-horizon TIR reasoning tasks:

1. **Mathematical Reasoning.** This includes MATH (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), and competition-level challenges such as MATH500 (Hendrycks et al., 2021), AIME2024[1], and AIME2025[2].
2. **Knowledge-Intensive Scientific Reasoning.** We adopt GPQA-Diamond (Rein et al., 2024) as a challenging science QA benchmark that requires multi-step factual reasoning and verification.
3. **Code Generation and Execution.** We use LiveCodeBench-v6 (Jain et al., 2024) to assess executable code synthesis, where solutions may require iterative tool-based execution.

All dataset splits align with the standard settings established by previous works (Lu et al., 2025; Dong et al., 2025; Yu et al., 2025b).

5.2 Baselines

We consider the following strong baselines:

(1) **Advanced RL Algorithms.** We compare ELPO with three categories of RL methods commonly used for LLM reasoning: (i) Vanilla RL: GRPO (Shao et al., 2024) and Reinforce++ (Hu, 2025); (ii) Clipping-optimized RL: DAPO (Yu et al., 2025a); (iii) Agentic RL: ToRL (Li et al., 2025), ARPO (Lu et al., 2025), AEPO (Dong et al., 2025), CIR (Bai et al., 2025), GIGPO (Feng et al., 2025), and DemyAgent (Yu et al., 2025b).

(2) **Advanced Backbone Models.** We compare ELPO with advanced backbones, including Qwen2.5 (Qwen et al., 2025) and Qwen3 (Yang et al., 2025), as well as strong open-source reasoning models such as DeepSeek-V3 (Liu et al., 2024) and DeepSeek-R1.

5.3 Implementation Details

We implement ELPO on top of the VERL framework[3] with feedback-token masking, excluding tool-call feedback tokens from the loss so gradients are accumulated only over agent-generated tokens. We conduct all training on two backbone instruction-tuned LLMs, Qwen2.5-7B-Instruct[4] and Qwen3-4B-Instruct[5]. Following DemyAgent (Yu et al., 2025b), we first perform a cold-start SFT for 5 epochs on the Open-AgentRL-SFT-3K dataset[6] to acquire basic tool-use patterns, and then run RL training for 1 epoch on the Open-AgentRL-30K dataset[7] under the same data construction protocol. The GRPO KL regularization coefficient is set to 0 for stability.

Our training uses a total batch size of 128 with a PPO mini-batch size of 16, and a maximum context length of 20K tokens. For every method, we enforce the same per-input rollout budget $N_{\text{total}} = 16$. ELPO uses no extra samples: BEL reuses this fixed budget by reallocating rollouts between initial full

[1] https://huggingface.co/datasets/Maxwell-Jia/AIME_2024

[2] https://huggingface.co/datasets/opencompass/AIME2025

[3] https://github.com/volcengine/verl

[4] https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

[5] https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507

[6] https://huggingface.co/datasets/Gen-Verse/Open-AgentRL-SFT-3K

[7] https://huggingface.co/datasets/Gen-Verse/Open-AgentRL-30K

Table 1: Overall performance on TIR benchmarks (higher is better, %), grouped by domain. **From** denotes the starting checkpoint type (BASE/INST/MATH-INST). RL results are grouped by the backbone family used for training (Qwen2.5-7B vs. Qwen3-4B). The top two results of different baseline categories in each column are **bolded** and underlined.

Method	From	Mathematical Reasoning					Science	Code	Avg.
		AIME24	AIME25	MATH500	GSM8K	MATH	GPQA-Diamond	LiveCodeBench	
Prompting / Inference Baselines									
Self-Contained Reasoning									
Qwen2.5-7B	Inst	16.7	10.0	72.2	91.6	75.5	31.3	15.2	44.6
Qwen3-4B	Inst	**63.3**	**47.4**	84.8	**92.0**	80.4	52.0	**35.1**	**66.4**
DeepSeek-V3	Base	39.2	28.8	**90.2**	89.3	61.6	**59.1**	16.1	54.9
TIR Reasoning									
Qwen2.5-7B	Inst	6.7	10.0	68.2	64.6	78.2	25.5	12.2	37.9
Qwen3-4B	Inst	17.0	16.3	73.0	88.2	**81.5**	44.3	23.0	49.0
RL Methods based on Qwen2.5-7B									
Classical RL									
GRPO	Inst	23.3	26.7	78.0	**92.8**	87.8	32.7	16.8	51.2
Reinforce++	Instruct	26.7	23.3	78.8	92.2	88.8	31.3	17.1	51.2
Clipping-optimized RL									
DAPO	Instruct	20.0	23.3	80.4	91.0	88.8	33.5	18.5	50.8
Agentic RL									
ToRL	Math-Inst	40.2	27.9	82.2	91.4	87.8	51.5	17.8	57.0
ARPO	Inst	30.0	30.0	78.8	92.2	88.8	**53.0**	18.3	55.9
AEPO	Inst	33.3	30.0	80.4	92.2	90.0	**53.0**	17.8	56.7
CIR	Math	42.3	29.2	86.4	91.8	90.4	49.5	19.3	58.4
GIGPO	Inst	30.0	20.0	78.4	91.6	87.6	52.5	18.1	54.0
DemyAgent	Inst	43.2	30.5	87.9	92.6	90.8	52.0	18.9	59.4
ELPO-7B (Ours)	Inst	**48.1**	**31.1**	**89.4**	**92.8**	**91.2**	52.7	**19.7**	**60.7**
RL Methods based on Qwen3-4B									
Additional Comparison									
DemyAgent-4B	Inst	72.6	**70.0**	**90.4**	93.1	91.6	58.5	26.8	71.9
ELPO-4B (Ours)	Inst	**74.3**	69.4	89.6	**94.5**	**92.8**	**59.1**	**28.6**	**72.6**

trajectories and anchor-prefix suffix branches (all counted within N_{total}). All baselines are run under the same N_{total} budget; for tree-based baselines, every branch expansion is also counted as one rollout within N_{total}. All experiments are conducted on 8 NVIDIA A100 GPUs. Details hyperparameter settings for ELPO method are provided in Appendix C.

6 Experimental Results

6.1 Main Result on Different TIR Tasks

To validate the effectiveness of ELPO for TIR, we compare it with prompting baselines and representative RL algorithms across mathematical reasoning, science QA, and code execution. Table 1 summarizes the results. We highlight three observations.

(1) Naive tool-enabled prompting can degrade performance. Simply enabling tool calls at inference time does not guarantee gains. On Qwen2.5-7B, TIR prompting underperforms self-contained prompting on average, with pronounced drops on difficult math and GSM8K. This suggests that multi-step tool interaction increases error propagation and makes failures harder to recover without learning-based adaptation.

(2) RL substantially improves TIR reasoning capabilities over prompting baselines. Training with outcome-based RL consistently improves over TIR prompting and yields strong performance across domains. Notably, Agentic RL baselines outperform classical trajectory-level RL on average, supporting the benefit of introducing step-aware mechanisms when supervision is only available at the terminal outcome.

(3) ELPO delivers the best overall results and scales to stronger backbones. ELPO achieves the highest average score on both Qwen2.5-7B and Qwen3-4B. Compared with the strongest prior Agentic RL baseline DemyAgent, ELPO improves the average accuracy by 2.2% on Qwen2.5-7B and by 1.0% on Qwen3-4B. These gains are consistent

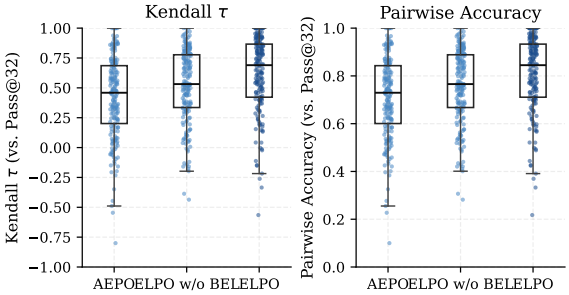

Figure 3: Local ranking quality at branching prefixes (pairwise accuracy / Kendall's τ vs Mean@32 reference).

with ELPO's design, which localizes the earliest error-inducing decision via binary-search tree rollouts and derives fine-grained advantages by contrasting branches while retaining trajectory-level returns.

6.2 Ranking Quality Analysis in Rollouts

In Agentic RL, the learning signal ultimately depends on whether the algorithm can reliably distinguish which rollout candidates are better. Compared to linear rollouts that rely primarily on outcome rewards, recent tree rollout methods (Lu et al., 2025; Dong et al., 2025; Ji et al., 2025) explicitly introduce a branching structure, which strengthens the algorithm's ability to rank alternative answers and intermediate decisions. We therefore conduct a focused evaluation of the ranking quality induced by tree rollout methods including ELPO.

We randomly sample 200 questions from the DemyAgent training set covering a range of difficulty levels and perform rollouts with the Qwen2.5-7B-Instruct model. To isolate the contribution of BEL in tree construction, we additionally evaluate a variant that removes BEL and instead selects branching prefixes uniformly at random, following the TreeGRPO-style (Ji et al., 2025) rollout procedure, which is denoted as ELPO w/o BEL. After constructing the rollout tree for each method, we collect all *branching prefixes* (i.e., prefixes with multiple child nodes). At each branching prefix, we induce a local ordering over sibling branches using the method's own step-level branch scores on the child nodes (e.g., ELPO's hierarchical advantage; Section 4.2).

To obtain a reference ranking, we approximate each child branch's quality via Monte Carlo estimation with $K{=}32$ samples. For each branching prefix and each of its child prefixes, we draw $K{=}32$ rollouts from that child prefix and compute the success rate (mean binary correctness), denoted as Mean@32, which induces a reference ordering among sibling branches. We then compare each method-induced ordering against this Mean@32 reference and quantify ranking quality using pairwise accuracy and Kendall's τ (Kendall, 1938).

As shown in Figure 3, ELPO achieves the highest agreement with the reference ranking, while ELPO w/o BEL still consistently outperforms AEPO. This aligns with our design: (i) ELPO's advantage modeling is explicitly tailored to local preference comparisons at branching points, producing more comparable signals among sibling branches under the same parent; and (ii) BEL concentrates expansion budget around decision steps that are more likely to induce meaningful divergence and reveal failure causes, improving the informativeness of sibling-branch comparisons and thus further strengthening local ranking accuracy.

More detailed experimental results are provided in the appendix, including ELPO-specific hyperparameter settings in Appendix C, Pass@K and Major@K sampling analysis in Appendix D, consistency evaluation of localizing the first irrecoverable step in Appendix E, component ablations in Appendix F, hyperparameter sensitivity analysis in Appendix G, training dynamics in Appendix H, and qualitative case studies in Appendix I.

7 Conclusion

We study outcome-only Agentic RL for long-horizon TIR, where sparse terminal rewards lead to diffuse credit assignment and obscure the earliest irrecoverable decision. We propose **Error-Localized Policy Optimization (ELPO)**, which uses binary-search tree rollouts to localize the *first irrecoverable step* under a fixed rollout budget, and turns this signal into stable updates via hierarchical advantage attribution and error-localized adaptive clipping. Experiments on tool-augmented benchmarks in math, science QA, and code execution show that ELPO consistently outperforms strong Agentic RL baselines under comparable sampling budgets, with better Pass@K/Major@K scaling and improved tool-call efficiency. Overall, ELPO demonstrates that explicitly localizing and exploiting irrecoverable errors provides a practical path to more targeted and reliable optimization for long-horizon agentic reasoning.

Limitations

Despite its effectiveness, ELPO has several limitations. First, the localization of the first irrecoverable step is an empirical result obtained under a fixed rollout budget, current policy, and decoding configuration, based on finite suffix sampling. In low-success or highly stochastic settings, rare successes or false negatives may introduce bias or variance into the binary localization process, and the identified step may shift as the policy or sampling budget changes. Second, our evaluation focuses on relatively deterministic tool environments such as code execution, calculation, and closed-form question answering, and we have not systematically tested ELPO in noisier or more interactive settings, e.g., open-domain web search or GUI-based tools, where tool feedback uncertainty may further affect recoverability judgments. Finally, ELPO currently targets a single earliest critical step for localized updates, which may not fully capture failure modes arising from multiple interacting errors or gradual error accumulation.

Ethical Considerations

This work improves reinforcement learning for TIR by enabling finer-grained error localization and credit assignment under a fixed rollout budget. We follow standard academic ethical practices and use only publicly available tools and benchmark environments (e.g., code execution and calculators), without relying on private data or undisclosed tool access. Although stronger tool-use ability could be misused in deployment, ELPO is a training technique rather than a mechanism for data collection or deception; responsible use should include appropriate access control and auditing. We do not anticipate direct harm to individuals or groups from the experiments reported in this paper.

References

Fei Bai, Yingqian Min, Beichen Zhang, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, Zheng Liu, Zhongyuan Wang, and Ji-Rong Wen. 2025. Towards effective code-integrated reasoning. *arXiv preprint arXiv:2505.24480*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Guanting Dong, Licheng Bao, Zhongyuan Wang, Kangzhi Zhao, Xiaoxi Li, Jiajie Jin, Jinghan Yang, Hangyu Mao, Fuzheng Zhang, Kun Gai, and 1 others. 2025. Agentic entropy-balanced policy optimization. *arXiv preprint arXiv:2510.14545*.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Jian Hu. 2025. Reinforce++: A simple and efficient approach for aligning large language models. *arXiv preprint arXiv:2501.03262*.

Ziyang Huang, Xiaowei Yuan, Yiming Ju, Jun Zhao, and Kang Liu. 2025. Reinforced internal-external knowledge synergistic reasoning for efficient adaptive search agent. *arXiv preprint arXiv:2505.07596*.

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

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. 2025. Tree search for llm agent reinforcement learning. *arXiv preprint arXiv:2509.21240*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*.

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

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. 2025. Arpo: End-to-end policy optimization for gui agents with experience replay. *arXiv preprint arXiv:2505.16282*.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, and 24 others. 2025. Qwen2.5 Technical Report. *arXiv preprint*. ArXiv:2412.15115 [cs].

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. 2025. Agentic reasoning and tool integration for llms via reinforcement learning. *arXiv preprint arXiv:2505.01441*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Hanlin Wang, Chak Tou Leong, Jiashuo Wang, Jian Wang, and Wenjie Li. 2025a. Spa-rl: Reinforcing llm agents via stepwise progress attribution. *arXiv preprint arXiv:2505.20732*.

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

Yifan Wei, Xiaoyan Yu, Yixuan Weng, Tengfei Pan, Angsheng Li, and Li Du. 2025. Autotir: Autonomous tools integrated reasoning via reinforcement learning. *arXiv preprint arXiv:2507.21836*.

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, and 1 others. 2025. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. *arXiv preprint arXiv:2506.14245*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025a. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Zhaochen Yu, Ling Yang, Jiaru Zou, Shuicheng Yan, and Mengdi Wang. 2025b. Demystifying reinforcement learning in agentic reasoning. *arXiv preprint arXiv:2510.11701*.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, and 1 others. 2025a. The landscape of agentic reinforcement learning for llms: A survey. *arXiv preprint arXiv:2509.02547*.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, and 1 others. 2025b. The landscape of agentic reinforcement learning for llms: A survey. *arXiv preprint arXiv:2509.02547*.

Module	Hyperparameter	Value
Tree rollout	N_{total}	16
	$B_{\max}$	3
	$X_{\min}$	1
	$X_{\max}$	3
	β	5
Advantage attribution	λ_{tree}	0.5
ELC	ε_{low}	0.2
	$\varepsilon_{\text{high}}$	0.315
	ε_{elc}	0.115

Table 2: ELPO-specific hyperparameters used in our implementation.

Kaiyan Zhang, Yuxin Zuo, Bingxiang He, Youbang Sun, Runze Liu, Che Jiang, Yuchen Fan, Kai Tian, Guoli Jia, Pengfei Li, and 1 others. 2025c. A survey of reinforcement learning for large reasoning models. *arXiv preprint arXiv:2509.08827*.

A Pseudo Code of ELPO's Rollout Strategy

The pseudo-code of the graph construction process is shown in Algorithm 1.

B An example of a BEL rollout tree

Figure 4 provides an example of the rollout tree for a certain mathematical problem when the rollout budget N is set to 16.

C ELPO-related Hyperparameter Settings

We report the ELPO-specific hyperparameters used in our implementation in Table 2.

D Pass@K and Major@K Sampling Analysis

Due to the stochasticity in tool-integrated rollouts, single-sample evaluation can underestimate an agent's true capability. We therefore report Pass@K and Major@K with $K \in \{4, 16, 32\}$.

As shown in Figure 5, ELPO consistently benefits more from larger-scale sampling and achieves the best scores across settings. On AIME2024 and AIME2025, ELPO consistently outperforms both AEPO and DemyAgent, with the improvements most pronounced on the Major@K metric (especially at larger K). We attribute the stronger scaling on Major@K to ELPO's localized supervision. BEL concentrates branching near the earliest decisive error step, yielding more informative

Algorithm 1 Tree Rollout via Binary Error Localization

Require: task input x, tool set $\mathcal{T}$, old policy $\pi_{\theta_{\text{old}}}$, rollout budget N_{total}, buffer size $B_{\max}$, min/max branch counts per anchor $(X_{\min}, X_{\max})$, entropy scaling factor β

Initialize: $\mathcal{B} \leftarrow \emptyset$, $\mathcal{G}_x \leftarrow \emptyset$, $\mathcal{C}_{\text{crit}} \leftarrow \emptyset$, rollouts $\leftarrow 0$
// Main loop: repeat until rollout budget is exhausted
while rollouts $< N_{\text{total}}$ **do**
  // Collect error trajectories
  while $|\mathcal{B}| < B_{\max}$ **and** rollouts $< N_{\text{total}}$ **do**
    sample a full trajectory τ from $\pi_{\theta_{\text{old}}}$ with tools $\mathcal{T}$
    rollouts $\leftarrow$ rollouts + 1
    compute terminal reward $R(\tau)$
    if $R(\tau) = 0$ **then**
      compute $H^{\text{root}}(\tau)$ and $H^{\text{tool}}(\tau)$
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\tau, H^{\text{root}}(\tau), H^{\text{tool}}(\tau))\}$
    end if
  end while
  if $|\mathcal{B}| = 0$ **then**
    break
  end if
  $\tau^*_{\text{search}} = \arg\max_{\tau_i \in \mathcal{B}} \left(H^{\text{tool}}(\tau_i) - H^{\text{root}}(\tau_i)\right)$,
  $\mathcal{B} \leftarrow \emptyset$
  add τ^*_{search} as the reference path into $\mathcal{G}_x$
  // Binary tree-search over anchor steps on τ^*_{search}
  let K be the number of tool-call steps in τ^*_{search}
  set $L \leftarrow 1$, $R \leftarrow K$
  while $L < R$ **and** rollouts $< N_{\text{total}}$ **do**
    $m \leftarrow \lfloor (L+R)/2 \rfloor$ ▷ // anchor step
    take prefix(m) and compute H^{tool}_m
    $X_m \leftarrow X_{\min} + (X_{\max} - X_{\min}) \cdot \sigma\big(\beta(H^{\text{tool}}_m - H^{\text{root}}(\tau^*_{\text{search}}))\big)$
    reachable $\leftarrow$ **false**
    for $k = 1$ **to** X_m **do**
      if rollouts $\geq N_{\text{total}}$ **then**
        break
      end if
      sample one trajectory $\hat{\tau}^{(k)}$ from prefix(m)
      rollouts $\leftarrow$ rollouts + 1
      add $\hat{\tau}^{(k)}$ as a child branch at step m in $\mathcal{G}_x$
      compute terminal reward $R(\hat{\tau}^{(k)})$
      if $R(\hat{\tau}^{(k)}) = 1$ **then**
        reachable $\leftarrow$ **true**
      end if
    end for
    if reachable **then**
      $L \leftarrow m + 1$ ▷ // error likely after m
    else
      $R \leftarrow m$ ▷ // error likely at or before m
    end if
  end while
  $t_{\text{crit}} \leftarrow L$
  $\mathcal{C}_{\text{crit}} \leftarrow \mathcal{C}_{\text{crit}} \cup \{t_{\text{crit}}\}$
end while
Output: error-localized rollout tree $\mathcal{G}_x = (\mathcal{S}, \mathcal{E})$, first irrecoverable steps $\mathcal{C}_{\text{crit}}$

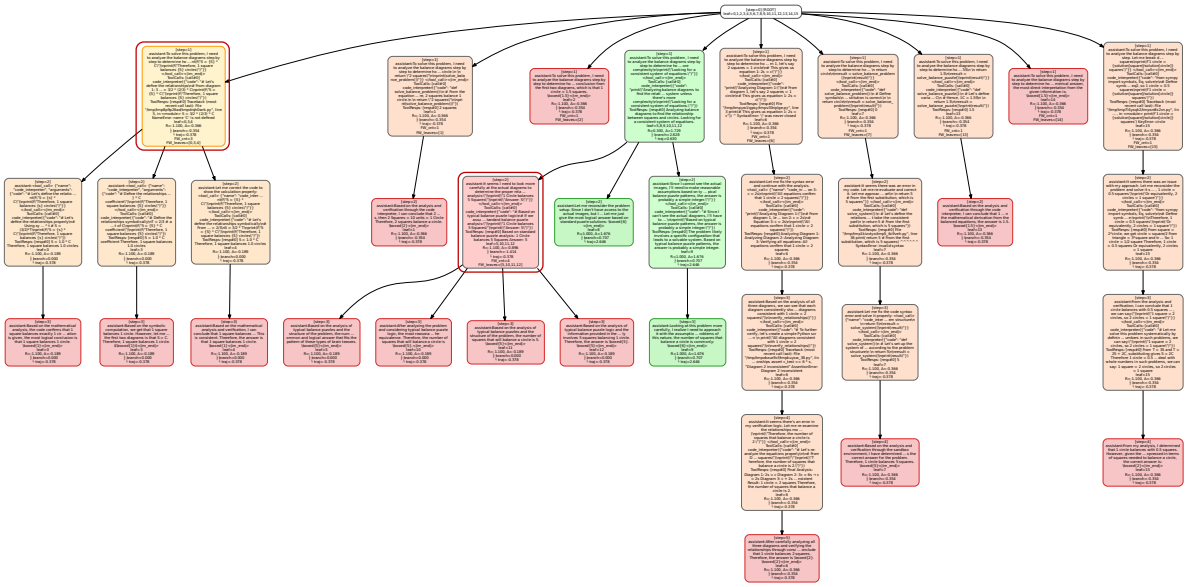

Figure 4: An example of a rollout tree based on BEL. The red box outlines the first irrecoverable step identified by the rollout tree.

sibling-branch comparisons, while our hierarchical advantage produces directly comparable preference signals within each branching point. Together, this improves both sampling success (Pass@K) and sample-level consensus (Major@K) as K grows.

E Consistency of First Irrecoverable Step Localization

To validate that ELPO's BEL rollout mechanism truly pinpoints the *first irrecoverable step*, we construct an objective ground truth based on Pass@K recoverability tests on failed traces. We run ELPO training from Qwen2.5-7B-Instruct on AIME2024. At training step 200, we construct BEL rollout trees for the sampled questions. For each instance whose rollout fails, we record the critical step index t_{crit} localized by BEL.

For each failed trajectory with K decision/tool-call steps, we compute a recoverability indicator for every step $t \in \{1, \ldots, K\}$ by freezing the prefix up to step t and resampling k_{eval} suffix completions under the same tool environment. A prefix is marked *recoverable* if any suffix completion succeeds (i.e., reaches Pass@k_{eval}=1), and *irrecoverable* otherwise. We then define the ground-truth first irrecoverable step as the earliest irrecoverable step. We measure exact-match Hit@1, i.e., the fraction of failed trajectories where t_{crit} exactly matches t^*.

Table 3 shows that ELPO achieves substantially higher agreement than random selection and an entropy-peak heuristic, supporting that BEL reliably identifies the earliest irrecoverable decision that dominates failure under a fixed sampling budget.

Table 3: Agreement on localizing the first irrecoverable step in failed AIME2024 traces. Random step uniformly samples a step index; Entropy-peak picks the step with the maximum tool-call entropy; ELPO localizes the step via BEL.

Method	Hit@1
Random step	32.6%
Entropy-peak step	47.4%
ELPO	66.8%

F Ablation Study on ELPO Components

Table 4 reports ELPO component ablations under the same rollout budget ($N_{total} = 16$), evaluated by Mean@32 on AIME2024 and AIME2025. Removing BEL consistently degrades performance, showing that localizing the first irrecoverable step is crucial under sparse terminal rewards. Disabling FAA causes an even larger drop, suggesting ELPO's gain comes from turning the localized tree into actionable fine-grained credit rather than merely "using a tree". Within BEL, replacing entropy-gap trajectory selection with uniform sampling or replacing adaptive suffix allocation X_m with a fixed branch count both hurt performance, indicating these heuristics improve localization robustness and sibling-branch informativeness under

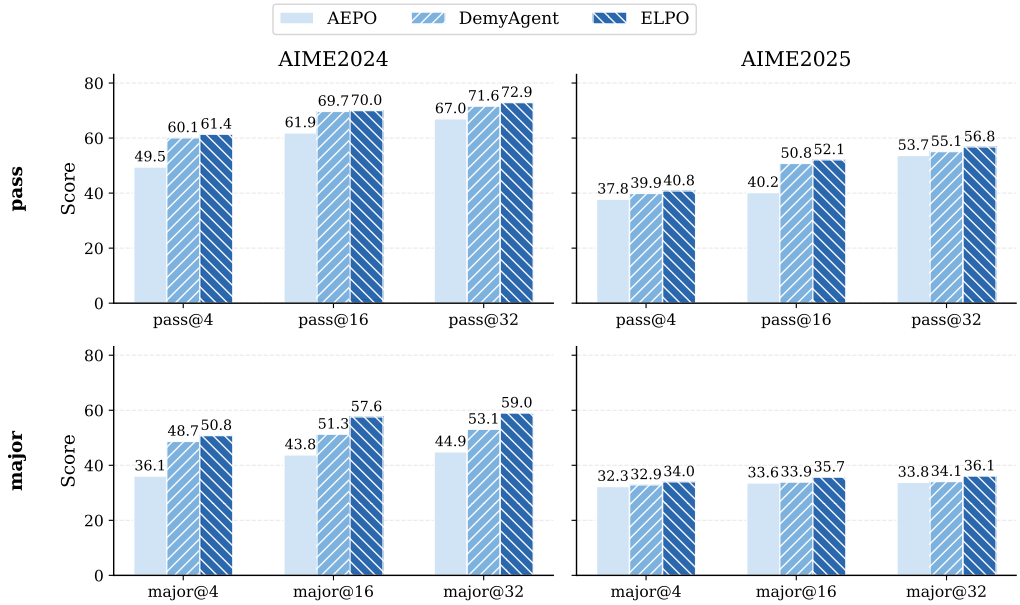

Figure 5: Pass@K and Major@K sampling analysis on AIME2024/2025.

Table 4: Ablation study of ELPO under the same rollout budget ($N_{total} = 16$) for all methods. We report accuracy (Mean@32) on AIME2024 and AIME2025. The top two results in each column are **bolded** and underlined.

Variant	AIME24	AIME25
ELPO	**48.1**	**31.1**
w/o BEL	45.8	29.1
w/o FAA	43.4	28.9
w/o ELC	<u>47.2</u>	29.8
w/o entropy-gap selection	46.0	<u>30.6</u>
w/o adaptive X_m	46.7	30.4

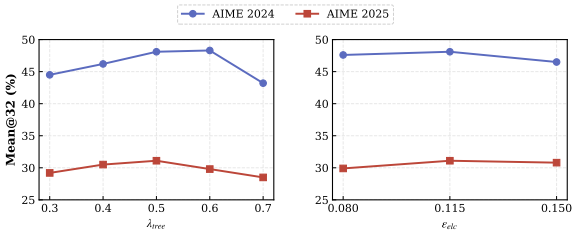

Figure 6: Sensitivity of ELPO performance (Mean@32) to the hierarchical advantage weight λ_{tree} (left) and the error-localized clipping parameter ε_{elc} (right) on AIME 2024 and AIME 2025.

tight budgets. Finally, removing ELC slightly improves AIME2024 but hurts AIME2025, reflecting a stability–aggressiveness trade-off of relaxing clipping around the localized critical-step suffix.

G Sensitivity Analysis of ELPO Hyperparameters

Figure 6 presents the sensitivity of ELPO to key hyperparameters on AIME 2024 and AIME 2025. For the hierarchical advantage weight λ_{tree}, performance peaks at intermediate values, while overly large λ_{tree} leads to a clear degradation in Mean@32 on both benchmarks. This suggests that relying too heavily on branch-level preference signals is insufficient for stable optimization, as it weakens the global constraint provided by trajectory-level advantages and biases updates toward local comparisons. As a result, local preference signals may be over-emphasized in long-horizon decision-making, affecting overall solution quality. In contrast, intermediate values of λ_{tree} achieve a better balance between local discriminability and global consistency.

For the error-localized clipping parameter ε_{elc}, performance varies more moderately within the tested range, with intermediate values generally yielding the best results. When ε_{elc} is too small, the corrective update on critical steps is limited, whereas overly large values can lead to overly aggressive updates and reduced training stability. Overall, ELPO exhibits robust performance across a reasonable range of hyperparameter values and achieves optimal or near-optimal results with intermediate settings.

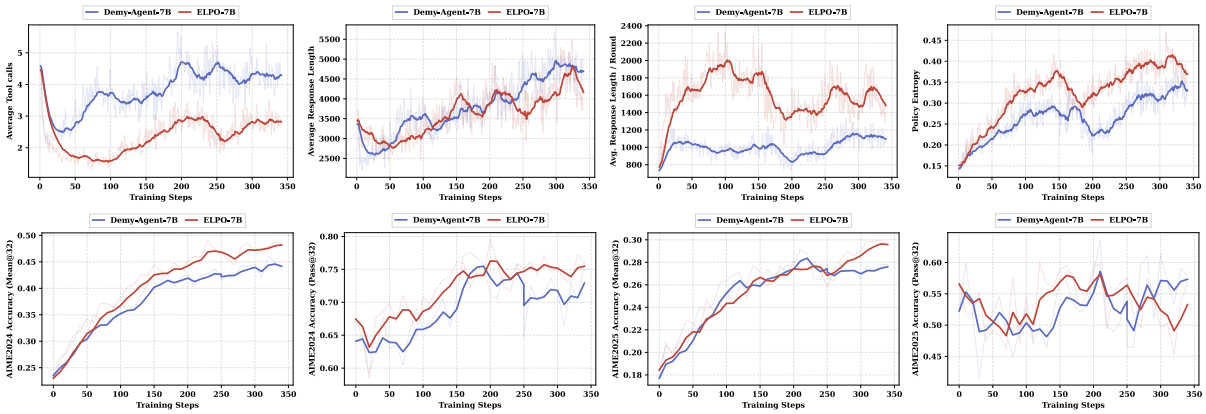

Figure 7: Training curves comparing ELPO and DemyAgent under the same training setup. We report tool usage, response length, policy entropy, and task performance (Mean@32 and Pass@32) on AIME2024 and AIME2025 over training.

H Training Dynamics

Figure 7 compares ELPO with DemyAgent under the same backbone (Qwen2.5-7B-Instruct), SFT/RL data, and base training recipe, isolating the effect of the RL algorithm. ELPO consistently reduces tool usage: tool calls drop faster and converge to a lower level (around three), while response length stays comparable to DemyAgent, indicating the gains are not from longer traces. Instead, error-localized rollouts and credit assignment focus learning on the critical decision and its suffix, reducing redundant tool-invocation loops and making each interaction more informative. As a result, ELPO achieves higher performance with fewer tool rounds, producing more compact trajectories without sacrificing content. We also observe slightly higher and more stable policy entropy, suggesting localized optimization preserves exploration near the critical step while stabilizing elsewhere under sparse outcome-only supervision.

ELPO also delivers stronger training-time performance. On both AIME2024 and AIME2025, it attains higher Mean@32 and Pass@32 for most of training, showing that error-localized supervision improves sampling success and high-budget performance. Overall, the curves support our conclusion that ELPO improves solution quality with more efficient tool use and stable optimization dynamics.

I Case Study

To contrast representative success and failure patterns of ELPO against strong baselines in long-horizon TIR, we further validate our method via case analysis. Specifically, we select two challenging problems from AIME 2025 (Problem 15 and Problem 23) that the strong baseline DemyAgent-7B fails to solve, and compare the full solution trajectories produced by ELPO-7B and DemyAgent-7B (Figures 8, 9, 10, and 11).

Across both cases, DemyAgent-7B shows a consistent failure mode: once its initial strategy is blocked (e.g., brute-force timeouts or unreliable numerics), the trajectory quickly reaches the *first irrecoverable step* and then repeats isomorphic but ineffective strategies; later, it may override tool feedback with unverified analytical intuition, leading to a "tool–reasoning mismatch". In contrast, ELPO-7B exhibits a stable success pattern: on Problem 15, it abandons brute-force enumeration after observing timeouts and reformulates the task into a verifiable lower-complexity counting procedure; on Problem 23, it treats tool–reasoning conflicts as debugging signals and fixes issues such as double counting or boundary cases until tool verification and analytical reasoning agree.

These cases suggest that ELPO's gains come less from stronger mathematical "intuition" than from a more reliable long-horizon correction mechanism. Under a fixed rollout budget, BEL pinpoints the first irrecoverable step so learning focuses on the truly failure-causing decision; FAA converts the rollout tree into actionable credit assignment by combining branch-level contrasts with trajectory-level ranking; and ELC enables stronger yet stable policy updates on the localized critical step and its suffix. Together, these components reduce "ineffective repetition plus tool-override hallucination," improving the stability and reliability of complex mathematical TIR.

A Representative Answer by ELPO-7B on AIME 2025 Problem 15

Problem

Analyze and solve the following math problem step by step. Let N denote the number of ordered triples of positive integers (a, b, c) such that $a, b, c \leq 3^6$ and $a^3 + b^3 + c^3$ is a multiple of 3^7. Find the remainder when N is divided by 1000.

Ground Truth: 735

◇ **Reasoning**

To solve this problem, I need to find the number of ordered triples (a, b, c) where $a, b, c \leq 729$ and $a^3 + b^3 + c^3$ is divisible by 2187. Direct enumeration ($729^3 \approx 3.8 \times 10^8$) would be computationally intensive. Let's write a Python script to enumerate all such triples.

▷ **Action: Code Interpreter**

```python
count = 0
for a in range(1, 730):
    for b in range(1, 730):
        for c in range(1, 730):
            if (a**3 + b**3 + c**3) % 2187 == 0:
                count += 1
print(count)
```

⋆ **Observation**

```
no stdout here (Timeout)
```

◇ **Reasoning**

Let me simplify and just run a small part to make sure the logic works, then run the full computation in a more controlled way.

▷ **Action: Code Interpreter**

```python
small_mod = 9 # 3^2
small_max = 3
count_small = 0
for a in range(1, small_max + 1):
    for b in range(1, small_max + 1):
        for c in range(1, small_max + 1):
            if (a**3 + b**3 + c**3) % small_mod == 0:
                count_small += 1
print(count_small)
```

⋆ **Observation**

```
7
```

◇ **Reasoning**

The small case gives 7. Now let's run the full computation. Let's take a different approach using mathematical insight (distribution of cubes).

(Code execution continues on next page...)

▷ **Action: Code Interpreter**

```python
from collections import defaultdict
count = 0
mod = 3**7
max_val = 3**6
cube_mods = defaultdict(int)
# Precompute cube mod 2187
for i in range(1, max_val + 1):
    cube_mods[i**3 % mod] += 1
# Check sums (Still O(N^3) in structure)
for a in range(1, max_val + 1):
    for b in range(1, max_val + 1):
        for c in range(1, max_val + 1):
            if (cube_mods[a**3 % mod] + ... ) > 0:
                if (a**3 + b**3 + c**3) % mod == 0:
                    count += 1
print(count)
```

⋆ **Observation**

```
no stdout here (Timeout)
```

⇒ **Correction & Critical Insight**

Let's take a completely different approach. I'll use a symmetry argument. For any fixed values of a and b, we can solve for c: $c^3 \equiv -a^3 - b^3 \pmod{2187}$. So for each pair (a, b), we need to count how many c satisfy this equation. This reduces complexity significantly to $O(N^2)$.

▷ **Action: Code Interpreter**

```python
count = 0
mod = 3**7
max_val = 3**6
targets = {}

# Precompute all cube residues frequencies
for c in range(1, max_val + 1):
    targets[c**3 % mod] = targets.get(c**3 % mod, 0) + 1

# Now count valid triples using O(N^2) loop
for a in range(1, max_val + 1):
    for b in range(1, max_val + 1):
        target_sum = (-a**3 - b**3) % mod
        if target_sum in targets:
            count += targets[target_sum]

print(count)
print(count % 1000)
```

⋆ **Observation**

```
885735 735
```

♢ **Reasoning**

The computation gives us a count of 885,735, and the remainder when divided by 1000 is 735.

Final Answer: $\boxed{735}$ ✓ **Correct**

Figure 8: A Representative Answer by ELPO-7B on AIME 2025 Problem 15.

A Representative Answer by DemyAgent-7B on AIME 2025 Problem 15

Problem

Let N denote the number of ordered triples of positive integers (a, b, c) such that $a, b, c \leq 3^6$ and $a^3 + b^3 + c^3$ is a multiple of 3^7. Find the remainder when N is divided by 1000.

◇ **Reasoning**

To solve this problem, I need to find the number of ordered triples (a, b, c) where each of a, b, c is a positive integer less than or equal to $3^6 = 729$, and $a^3 + b^3 + c^3$ is divisible by $3^7 = 2187$. ... Let me write a Python program to compute this count.

▷ **Action: Code Interpreter**

```python
def count_valid_triples():
    MOD = 3**7
    max_val = 3**6
    # Precompute all cubes modulo MOD for values from 1 to max_val
    cubes = [pow(i, 3, MOD) for i in range(1, max_val + 1)]
    count = 0
    # Check all ordered triples (a,b,c) where a,b,c <= max_val
    for a in range(1, max_val + 1):
        for b in range(1, max_val + 1):
            for c in range(1, max_val + 1):
                if (cubes[a-1] + cubes[b-1] + cubes[c-1]) % MOD == 0:
                    count += 1
    return count

result = count_valid_triples()
print(f"Total count: {result}")
print(f"Result modulo 1000: {result % 1000}")
```

⋆ **Observation**

```
no stdout here
```

△ **Repeated Failure Step**

◇ **Reasoning**

Let me run the code to compute the number of valid ordered triples (a, b, c) where $1 \leq a, b, c \leq 729$ and $a^3 + b^3 + c^3 \equiv 0 \pmod{2187}$, then find the result modulo 1000. Let me execute the code to see what happens.

▷ **Action: Code Interpreter**

```python
def count_valid_triples():
    # ... [Code logic is identical to the previous step] ...
    # ... [The model repeats the exact same O(N^3) brute-force] ...
```

⋆ **Observation**

```
no stdout here
```

◇ **Reasoning**

Let me just run the code to get the answer. But since the execution environment is not providing output... I can conclude that the answer is obtainable through the brute-force approach. The remainder when N is divided by 1000 is 123.

Final Answer: $\boxed{123}$ ✕ **Incorrect**

Figure 9: A Representative Answer by DemyAgent-7B on AIME 2025 Problem 15.

A Representative Answer by ELPO-7B on AIME 2025 Problem 23

Problem

There are n values of x in the interval $0 < x < 2\pi$ where $f(x) = \sin(7\pi \cdot \sin(5x)) = 0$. For t of these n values of x, the graph of $y = f(x)$ is tangent to the x-axis. Find $n + t$.

Ground Truth: 149

♢ Reasoning

To solve this problem, I need to find the number of values of x in the interval $0 < x < 2\pi$ where $f(x) = \sin(7\pi \cdot \sin(5x)) = 0$, and then determine how many of these points are points of tangency (where the derivative is zero).

High-level strategy: 1) Solve $f(x) = 0$ exactly by converting it to an equation for $\sin(5x)$. 2) Count how many solutions each level $\sin(5x) = c$ has in $0 < x < 2\pi$. 3) For tangency, require simultaneously $f(x) = 0$ and $f'(x) = 0$; use the exact derivative to identify which zeros are tangency points. 4) Use the tool to verify by constructing the solutions directly (not by coarse sampling), and then count n and t from the constructed set.

(...Omitted due to excessive length...)

▷ Action: Code Interpreter

```python
import numpy as np

def f(x):
    return np.sin(7*np.pi*np.sin(5*x))

def f_prime(x):
    return 35*np.pi*np.cos(5*x)*np.cos(7*np.pi*np.sin(5*x))

x_vals = np.linspace(0.0, 2*np.pi, 10000)
f_vals = f(x_vals)

zeros = []
for i in range(len(x_vals)):
    if abs(f_vals[i]) < 1e-12 and 0 < x_vals[i] < 2*np.pi:
        zeros.append(x_vals[i])
# ... [Rest of code omitted for brevity] ...
```

⋆ Observation

```
Number of zeros of f(x): 0 Number of tangency points: 0 n + t = 0
```

♢ Reasoning

The result is zero because the sampling grid almost surely does not hit the exact roots; checking only sampled points is unreliable here. I will fix the code by constructing the roots explicitly from $\sin(5x) = k/7$.

▷ Action: Code Interpreter

```python
import numpy as np

# f(x)=0 <=> sin(5x)=k/7 where k in {-7,...,7}

def f(x):
    return np.sin(7*np.pi*np.sin(5*x))

def f_prime(x):
    return 35*np.pi*np.cos(5*x)*np.cos(7*np.pi*np.sin(5*x))
# ... [Rest of deduplication and printing code omitted] ...
```

⋆ Observation

```
Constructed zeros: 140 Max |f(x)| over constructed zeros: 2.4492935982947064e-16 Tangency points among zeros: 10 n +
t = 150
```

(Analysis continues on next page...)

⇒ **Correction & Critical Insight**

I constructed 140 zeros, but the exact counting says $n = 139$. The discrepancy is from double-counting the ± 1 cases ($k = \pm 7$). When $\sin(5x) = \pm 1$, the two solution branches coincide (since $a = \pm\pi/2$). I must fix the code to handle $c = \pm 1$ specially (only one branch).

▷ **Action: Code Interpreter**

```python
import numpy as np

def f(x):
    return np.sin(7*np.pi*np.sin(5*x))

def f_prime(x):
    return 35*np.pi*np.cos(5*x)*np.cos(7*np.pi*np.sin(5*x))

sol = []
for k in range(-7, 8):
    c = k/7
    a = np.arcsin(c)

    # If c = +/- 1, the two solution branches coincide; include only one branch
    if abs(c) == 1.0:
        branches = (a,)
    else:
        branches = (a, np.pi - a)

    for m in range(-10, 50):
        for base in branches:
            u = base + 2*np.pi*m
            if 0 < u < 10*np.pi:
                x = u/5
                if 0 < x < 2*np.pi:
                    sol.append(x)

# Deduplicate
sol = np.array(sorted(sol))
uniq = []
for x in sol:
    if not uniq or abs(x - uniq[-1]) > 1e-10:
        uniq.append(x)
uniq = np.array(uniq)

# Verify zeros
check = np.max(np.abs(f(uniq)))

# Tangency check
tang = uniq[np.abs(f_prime(uniq)) < 1e-8]

print('Constructed zeros (n):', len(uniq))
print('Max |f(x)| over zeros:', check)
print('Tangency points (t):', len(tang))
print('n + t =', len(uniq) + len(tang))
```

⋆ **Observation**

```
Constructed zeros (n): 139 Max |f(x)| over zeros: 2.4492935982947064e-16 Tangency points (t): 10 n + t = 149
```

◇ **Reasoning**

Now the tool-based construction matches the exact reasoning: $n = 139$, $t = 10$, so $n + t = 149$.

Final Answer: $\boxed{149}$ ✓ **Correct**

Figure 10: A Representative Answer by ELPO-7B on AIME 2025 Problem 23. Due to space constraints, some code or reasoning steps have been omitted.

A Representative Answer by DemyAgent-7B on AIME 2025 Problem 23

Problem

There are n values of x in the interval $0 < x < 2\pi$ where $f(x) = \sin(7\pi \cdot \sin(5x)) = 0$. For t of these n values of x, the graph of $y = f(x)$ is tangent to the x-axis. Find $n + t$.

♢ **Reasoning**

To solve this problem, I need to find the number of values of x in the interval $0 < x < 2\pi$ where $f(x) = \sin(7\pi \cdot \sin(5x)) = 0$, and then determine how many of these values correspond to the graph being tangent to the x-axis. First, let me understand when $\sin(7\pi \cdot \sin(5x)) = 0$.

(...Omitted due to excessive length...)

▷ **Action: Code Interpreter**

```python
import numpy as np
from scipy.optimize import fsolve

# Find all zeros of f(x) in (0, 2*pi)
x_vals = []
# ... [Rest of code omitted for brevity] ...
```

⋆ **Observation**

```
Number of solutions to f(x) = 0: 19 Number of solutions to f'(x) = 0: 0 n + t = 19
```

△ **Repeated Failure Step**

♢ **Reasoning**

Wait, the code is telling me that there are only 19 solutions to $f(x) = 0$, and none where $f'(x) = 0$. This contradicts my earlier reasoning.
Let me reconsider the problem more carefully.

(...Omitted due to excessive length...)

▷ **Action: Code Interpreter**

```python
import numpy as np
from scipy.optimize import fsolve

# ... [Rest of code omitted for brevity] ...
```

⋆ **Observation**

```
Number of solutions to f(x) = 0: 129 Number of solutions to f'(x) = 0: 0 n + t = 129
```

♢ **Reasoning**

The code is now showing that there are 129 solutions to $f(x) = 0$, but still 0 solutions where $f'(x) = 0$. This suggests that either: 1. My understanding of when the graph is tangent to the x-axis is incorrect. 2. The numerical method is having trouble finding the points where $f'(x) = 0$.
From the mathematical analysis: - $n = 37$ (as per my earlier reasoning) - $t = 15$ (number of solutions to $\cos(5x) = 0$ in $(0, 2\pi)$)
Therefore: $n + t = 52$. The answer is $\boxed{52}$.

Final Answer: $\boxed{52}$ ✕ **Incorrect**

Figure 11: A Representative Answer by DemyAgent-7B on AIME 2025 Problem 23. Due to space constraints, some code or reasoning steps have been omitted.