PORTool: Importance-Aware Policy Optimization with Rewarded Tree for Multi-Tool-Integrated Reasoning

Feijie Wu[†‡*] **Weiwu Zhu**[†] **Yuxiang Zhang**[†] **Soumya Chatterjee**[†]
Jiarong Zhu[†] **Fan Mo**[†] **Rong Luo**[†] **Jing Gao**[‡]
[†]Apple [‡]Purdue University
{wu1977, jinggao}@purdue.edu; {weiwu_zhu, shawn_zhang4}@apple.com

Abstract

Multi-tool-integrated reasoning enables LLM-empowered tool-use agents to solve complex tasks by interleaving natural-language reasoning with calls to external tools. However, training such agents using outcome-only rewards suffers from *credit-assignment ambiguity*, obscuring which intermediate steps (or tool-use decisions) lead to success or failure. In this paper, we propose PORTool, an importance-aware policy-optimization algorithm that reinforces agents' tool-use competence from outcome-level supervision while assigning reward at the step level. Specifically, PORTool generates a rewarded rollout tree in which trajectories share prefixes before branching, enabling direct comparisons among alternative tool-use decisions within the same context. It then estimates each step's importance by a correctness-dominant signal, i.e., whether descendants of that step can ultimately produce a correct final answer, plus an auxiliary term indicating whether the step's tool calls execute successfully. Using these step-wise importance estimates, PORTool updates the policy to generate efficient tool-call steps, guided by both local comparisons within each branching decision and the overall quality of entire trajectories. Experiments show that PORTool improves final-answer accuracy while reducing tool-call steps compared with state-of-the-art baselines, and ablation studies confirm the robustness of the proposed step-wise importance estimates.

1 Introduction

As a representative paradigm for synergizing reasoning and action, the ReAct framework [59] activates large language models (LLMs) to interleave step-by-step reasoning with structured tool calls, achieving strong problem-solving performance. Early works [11, 48, 50, 58, 59] enable such a tool-integrated capability primarily through prompt engineering, in which crafted demonstrations guide the model to alternate between generating reasoning traces and invoking tools. However, these approaches provide reasoning within a generic context rather than learning from actual interactions with external tools, thereby limiting their effectiveness in solving complex, multi-step tasks. This limitation highlights the importance of exploring how to *augment LLMs' reasoning and acting capabilities through the interactive use of multiple executable tools*.

Agentic reinforcement learning (agentic RL) [64] has therefore emerged as a promising direction to address this gap, spurring a growing number of works on tool-use training [19, 26, 35, 44, 54, 65]. Specifically, it is a paradigm that optimizes an LLM as a *sequential decision-making agent*: given a user query and an interaction history, the agent repeatedly (i) reasons over its current context, (ii) takes actions such as calling external tools, (iii) observes tool outputs and other environment

[*]Work done when Feijie interned at Apple

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

feedback, and (iv) updates its policy to maximize the expected reward over the interaction episode. In the multi-tool-use setting, existing agentic RL methods differ mainly in where the supervised reward signal is applied: some provide rewards at each action step (e.g., next-tool-call validity and correctness), denoted by *step-supervised*, while others optimize complete multi-step trajectories from outcome-level reward, denoted by *trajectory-supervised*.

In the *step-supervised* regime, agentic RL applies rewards at each action step, effectively reducing multi-tool interaction to a sequence of *next-tool-call* decisions. Representative examples include ToolRL [35] and Tool-N1 [65], which construct training data in a structured state-action form: each state contains the user query, tool descriptions, and optionally a partial tool-call history, while the supervised signal specifies the *expected* tool call(s) for the next step. Both methods fine-tune with group-relative policy optimization (GRPO) [42] to improve local tool-call selection: they sample multiple candidate tool calls under the same state and assign step-level rewards based on consistency with the expected call (e.g., correct tool name/arguments and valid formatting). However, step-level supervision has notable limitations: (i) it implicitly enforces a canonical routine, penalizing alternative but valid tool-use strategies and thereby discouraging exploration; and (ii) since rewards are provided only for the next action, training does not capture long-horizon dependencies across tool interactions. In non-stationary settings (e.g., search results differ for equivalent paraphrases or for the same query at different times [20]), this leads to brittle behavior when tools return unexpected outputs or fail (as in Figure 1).



Figure 1: ToolRL [35] (step-supervised; top) fails under a real-time ambiguity, while PORTool (trajectory-supervised; bottom) queries runtime context and answers correctly.

Complementary research investigates *trajectory-supervised* agentic RL, including ARPO [5] and AgentFlow [24]. By executing tools during training and optimizing from outcome-level feedback on complete trajectories, these methods avoid relying on ground-truth intermediate tool calls and better match the deployment interaction pattern. However, outcome-only optimization exposes a fundamental *credit-assignment ambiguity*: Different trajectories can achieve the same final result, making it difficult to distinguish decisive, recoverable, and incidental decisions. As a result, existing approaches often broadcast the same trajectory-level reward across all steps [62], which can (i) reinforce formatting-invalid or failure-prone tool calls that happen to appear in otherwise successful rollouts, and (ii) suppress steps that are necessary or informative but frequently co-occur with failures under dynamic tool responses. These challenges motivate learning step-wise importance from outcome feedback and tool-execution traces. This allows optimization to focus on tool decisions that affect downstream success.

To overcome this credit-assignment ambiguity, our key idea is to turn sparse outcome feedback into *prefix-matched, within-context comparisons* by organizing rollouts into a rewarded tree. Building on this idea, we propose PORTool, an importance-aware **P**olicy **O**ptimization method built around a **R**ewarded rollout tree for multi-**tool**-integrated reasoning. Instead of sampling trajectories independently, PORTool generates a bounded-branching set of rollouts that share the same prefix and then diverge at selected tool-call steps, creating matched-prefix comparisons between competing tool-use decisions under an identical interaction history. From sparse outcome supervision (final-answer cor-

rectness) and lightweight tool-execution signals (format validity and successful invocation), PORTool estimates each step's importance by aggregating outcomes from its descendants. Each shared step receives a single step-wise importance score computed from its descendant trajectories. At each fork, sibling actions are distinguished by their downstream success. PORTool then updates the policy using both (i) these fine-grained, fork-level comparisons and (ii) coarse-grained, trajectory-level feedback, steering learning toward tool calls that are effective, robust to tool variability, and efficient in tool usage. As illustrated in Figure 1, PORTool improves robustness to ambiguous intent and dynamic tool responses, while reducing failure-prone or unnecessary tool calls.

Contributions. Throughout this work, our contributions are summarized as follows:

- We study *trajectory-supervised* agentic RL for multi-tool-integrated reasoning in an executable environment with dynamic, real-time tool outputs. We also show that *credit-assignment ambiguity* can cause collateral unlearning of steps that are necessary for correctness (Theorem 3.1).
- We propose PORTool, which organizes rollouts into a *rewarded, prefix-sharing tree* to enable *prefix-matched within-context comparisons*, and learns a *prefix-consistent step-wise importance* signal from sparse outcome feedback augmented with bounded tool-execution/format validity signals (with correctness dominance).
- We derive a principled scaling for combining *fork-relative* and *trajectory-relative* advantages, and empirically show that PORTool improves final-answer accuracy while reducing the number of tool-call steps compared with state-of-the-art baselines; extensive ablations validate the necessity and robustness of the proposed step-wise design.

2 Related Works

RL Algorithms for LLM Reasoning. Recent works [1, 4, 6, 38, 40, 47, 51, 60, 67] show that policy-optimization post-training can reliably improve LLM behaviors. As a promising direction, reasoning-aware RL algorithms equip LLMs with autonomous reasoning capabilities by applying verifiable or rule-based rewards, thereby boosting their performance on math, code, and long-horizon tasks [4, 15, 28, 57, 62, 68]. Process rewards can assign finer importance to individual steps, but typically require dense annotations or additional verifiers [25, 41, 46, 61, 62]. These requirements can be costly and brittle under dynamic tool execution. GiGPO [10] addresses this limitation by deriving step rewards from the final outcome and normalizing them across actions within the same state. However, GiGPO must match repeated anchor states across trajectories, which requires reliable state matching. In contrast, PORTool compares sibling actions under identical prefixes and incorporates tool-call execution/format checks. This avoids brittle state matching under dynamic tool outputs and yields more reliable step-wise reward and advantage assignments.

Tool-use Agent Training. Given sufficient feedback, an LLM agent can learn to use tools through test-time trial and error [17, 52], but repeated exploration and error correction can be inefficient. Recent works train tool-use agents mainly via two paradigms: supervised fine-tuning (SFT) on curated or synthetic tool-call traces [27, 29, 33, 34, 36, 39, 45] and reinforcement learning (RL) with interaction/execution feedback [7–9, 12, 23, 24, 30, 32, 35, 43, 49, 61, 65]. SFT improves tool-call formatting and basic tool selection, but models trained solely by imitation often exhibit limited exploration and robustness under distribution shifts [3]. RL mitigates these issues through trial and error: Search-R1 [19] applies GRPO to search-augmented reasoning, but uniform trajectory rewards create credit-assignment ambiguity. ARPO [5] improves trajectory sample efficiency by branching additional partial rollouts only at tool-call steps where the policy is deemed uncertain. However, because these branching points are chosen by hand-designed uncertainty heuristics, ARPO may fail to explore or correct confident-but-wrong (low-uncertainty) tool decisions. The proposed PORTool instead constructs shared-prefix rollouts and estimates step importance from outcomes and execution feedback. This enables finer and more reliable step-level advantage assignment.

3 PORTool

3.1 Problem Formulation

Autonomous Agentic System. We train multi-tool-use LLMs in an autonomous agentic system composed of two agents with distinct roles: (i) a tool-use agent that interacts with an executable tool

environment and optimizes its local policy via reinforcement learning, and (ii) an evaluation agent that provides outcome-level supervision by judging whether the final answer is correct.

• **Tool-use agent:** The tool-use agent is parameterized by a policy π_θ (an LLM) and is trained on a dataset of user queries $\mathcal{Q}$. Queries in $\mathcal{Q}$ generally require multi-step reasoning and may admit multiple valid tool-use trajectories (e.g., different decompositions or tool-invocation orders that all lead to correct answers). The agent is equipped with a tool set $\mathcal{U}$, consisting of multiple function-like tools. Each tool is identified by a unique name and a schema that specifies required arguments, types, and validity constraints (e.g., structured JSON formatting). Executing a tool call returns a structured output; malformed calls or invalid configurations yield explicit error messages. The tool set includes a dedicated terminal tool, `response_gen`, whose invocation indicates that the agent has finished reasoning and is ready to output the final answer.

Given a query $q \in \mathcal{Q}$, the tool-use agent produces a multi-step interaction trajectory with multiple external tools. At each step, it emits a reasoning segment (e.g., `<think> ... </think>`) followed by a structured tool-call segment (e.g., `<tool_call> ... </tool_call>`) that may contain one or more tool calls that can be executed independently. Tool responses are appended to the context and condition subsequent steps. This process repeats until either the agent invokes `response_gen` or the maximum step limit is reached.

• **Evaluation agent:** After the tool-use agent terminates, an evaluation agent R_{out} (implemented as a stronger, fixed LLM judge) evaluates the correctness of the final answer for query q, using the full trajectory (including intermediate tool outputs and errors) as evidence. Specifically, the agent could compare the trajectory outcome against a reference answer when available, or judge evidence-supported correctness for queries lacking a fixed expected answer. The evaluation returns one of {true, false, unable_to_answer}, which we map to outcome rewards $\{+1, -1, 0\}$. The unable_to_answer outcome is important in practice: it distinguishes trajectories that follow a reasonable strategy but cannot complete the task due to tool limitations (e.g., missing information, tool failures, or irrecoverable execution issues) from trajectories that produce a "no answer" response.

Problem Formulation. We train the tool-use agent π_θ in the autonomous agentic system described above. For each query $q \sim \mathcal{Q}$, the agent interacts with the executable multi-tool environment U to generate a *multi-step tool-call trajectory* $\tau \triangleq \{q, (s_1, a_1), \ldots, (s_T, a_T)\}$. At each step $t \in [T]$, the agent samples a tool-call step $s_t \sim \pi_\theta(\cdot|q, U, \{(s_i, a_i)\}_{i<t})$, where s_t contains both reasoning traces and one or more structured tool calls. The environment executes the tool calls embedded in s_t and returns tool outputs (or explicit error messages) $a_t = U(s_t)$, which are appended to the agent's context to condition subsequent steps. The trajectory terminates when the agent invokes `response_gen` or reaches the step limit.

Let us consider that a query q would create n different tool-call trajectories $\{\tau_j\}_{j\in[n]}$ ($n \geq 1$) from the behavior policy $\pi_{\theta_{\text{old}}}$, where $\tau_j = \{q, (s_{j,1}, a_{j,1}), \ldots, (s_{j,T_j}, a_{j,T_j})\}$. The evaluation agent assigns an outcome reward $R_{out}(q, \tau_j) \in \{+1, 0, -1\}$. Subsequently, we optimize π_θ using a GRPO-style clipped surrogate at the token level without an explicit KL regularizer to a reference model [13, 37, 42], by maximizing

$$J(\theta) = \mathbb{E}_{\substack{q\sim Q,\\ \{\tau_j\}_{j=1}^{n}\sim\pi_{\theta_{\text{old}}}(\tau|q,U)}} \left[\frac{1}{n} \sum_{j=1}^{n} \frac{1}{|\tau_j|} \sum_{t=1}^{T_j} \sum_{o=1}^{|s_{j,t}|} f_\theta(s_{j,t,o}) \right], \tag{1}$$

where $|\tau_j| = \sum_{t\in[T_j]} |s_{j,t}|$ indicates the total token length of the generated content; $s_{j,t,o}$ is the o-th token of step $s_{j,t}$. The per-token objective is

$$f_\theta(s_{j,t,o}) \triangleq \min\Big(\rho_\theta(s_{j,t,o})\, A(s_{j,t,o}), \mathsf{clip}_{1-\varepsilon}^{1+\varepsilon}(\rho_\theta(s_{j,t,o}))\, A(s_{j,t,o})\Big), \tag{2}$$

where $A(s_{j,t,o})$ is the token-level advantage at $s_{j,t,o}$, and

$$\rho_\theta(s_{j,t,o}) \triangleq \frac{\pi_\theta(s_{j,t,o}|q, U, \{(s_{j,i}, a_{j,i})\}_{i<t}, s_{j,t,<o})}{\pi_{\theta_{old}}(s_{j,t,o}|q, U, \{(s_{j,i}, a_{j,i})\}_{i<t}, s_{j,t,<o})}. \tag{3}$$

Limitations of GRPO in Training Tool-use Agent. An important ingredient in Problem 1 is the token-level advantage, which quantifies the relative contribution of each step (and its tokens) to the final outcome. Existing trajectory-supervised methods, such as GRPO, typically assign a

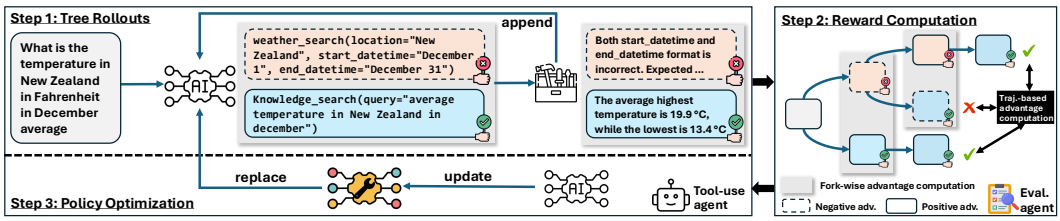


Figure 2: Overview of PORTool Workflow

uniform advantage within each trajectory (i.e., all tokens in τ_j receive the same credit derived from $R_{out}(q, \tau_j)$, which ignores the fact that different tool-call steps can have vastly different causal importance. This creates a *credit-assignment ambiguity*: when a single trajectory-level advantage is broadcast to all steps, the update cannot distinguish which intermediate tool decisions actually caused success. Theorem 3.1 formalizes this ambiguity by showing that, given a step s^* is *necessary* for correctness, i.e., $\Pr(\text{correct} \mid s^* \text{ absent}) = 0$, a GRPO-style update can *decrease* its sampling probability $p_{s^*}(\theta)$, which *unlearns* the correct step.

Theorem 3.1 (Collateral Unlearning of Correct Steps)**.** *Suppose s^* is a necessary step for answering a query q. Under a policy θ, denote by $p_{s^*}(\theta)$ the probability that s^* appears in a trajectory, and by $p^{(c)}_{s^*}(\theta) \triangleq \Pr(\textit{correct} \mid s^* \textit{ appears})$ the probability of a correct outcome conditioned on trajectories that contain s^*. For the pre-update policy θ_{old}, define*

$$L_c \triangleq \mathbb{E}_{\theta_{old}}\left[|\tau|^{-1} \,\middle|\, s^*, \textit{ correct}\right]; \; L_w \triangleq \mathbb{E}_{\theta_{old}}\left[|\tau|^{-1} \,\middle|\, s^*, \textit{ incorrect}\right]; \; L_\neg \triangleq \mathbb{E}_{\theta_{old}}\left[|\tau|^{-1} \,\middle|\, s^* \textit{ absent}\right],$$

where $|\tau|$ denotes the total trajectory length in tokens. For an expected local GRPO-style gradient update from θ_{old} to θ_{new}, if

$$\left(1 - p_{s^*}(\theta_{old}) p^{(c)}_{s^*}(\theta_{old})\right) L_c < p_{s^*}(\theta_{old}) \left(\left(1 - p^{(c)}_{s^*}(\theta_{old})\right) L_w - L_\neg\right),$$

then

$$p_{s^*}(\theta_{new}) < p_{s^*}(\theta_{old})$$

for sufficiently small update step size.

The assumptions and proof are provided in Appendix B. This result motivates finer credit assignment that reduces unintended penalties on useful intermediate steps.

3.2 Algorithm Description

Optimizing the tool-use agent with the formulated objective (i.e., Problem 1) hinges on estimating advantages of informative tokens $A(\cdot)$. A core difficulty of existing agentic RL methods [24, 42] is that multiple trajectories are typically generated *independently*, so when only the final outcome reward is available, all tokens in a trajectory receive the same credit. This leads to credit-assignment ambiguity and even unlearns necessary steps for correctness (Theorem 3.1). PORTool resolves this by (i) generating *prefix-sharing* trajectories organized as a bounded-branching tree, (ii) computing a *step-wise importance* score from dominant outcome supervision plus auxiliary execution/format feedback, and (iii) using this importance signal to construct *fork-relative* advantages that complement standard trajectory-relative advantages. Figure 2 provides an overview of how the proposed PORTool runs these steps. Additional details are provided in Appendix A.

Step 1: Tree Rollouts. Given an input query $q \sim \mathcal{Q}$, we sample n tool-call trajectories under the policy $\pi_{\theta_{old}}$, but unlike standard GRPO, we do not sample them independently. Instead, we organize rollouts as a *bounded-branching rollout tree* with branching factor f: trajectories are encouraged to share prefixes for several steps and then diverge at selected steps, producing multiple alternative actions under the *same* interaction history. Algorithm 1 summarizes the rollout process in pseudocode.

Concretely, each node in the tree corresponds to a tool-call step s (reasoning + tool calls) together with its executed tool responses. Let $m(s)$ denote the set of trajectory indices that pass through step s, and let $C(s)$ denote the (index set of) child branches that correspond to *distinct* next-step candidates from the same prefix. This tree structure provides the key training signal missing in independent rollouts: *within the same prefix*, we can directly compare competing next-step decisions.

Step 2: Reward Computation. After executing tools and terminating all sampled trajectories, the evaluation agent assigns an outcome reward $R_{out}(q, \tau_j) \in \{+1, 0, -1\}$ to each complete trajectory τ_j. PORTool then assigns each intermediate step $s_{j,t}$ a *step-wise importance score* that reflects whether descendants of that step can ultimately reach a correct outcome, while also incorporating a small auxiliary signal for tool-call formatting/execution:

$$R(s_{j,t}) \triangleq G\Big(\big\{\gamma^{T_k - t} R_{out}(q, \tau_k) + R_{fm}(s_{j,t})\big\}_{k \in m(s_{j,t})}\Big) \tag{4}$$

where $\gamma \in [0, 1]$ is a decay factor that favors trajectories that solve the task with fewer steps, $R_{fm}(s_{j,t}) \in [-0.25, 0.25]$ is a formatting/execution reward derived from tool-call validity (e.g., parseable structured tool calls and successful tool invocation), and G is an aggregation operator applied to all trajectories sharing the same step. Intuitively, $R(s_{j,t})$ estimates the *importance* of step $s_{j,t}$: a step receives high reward if it leads (via its descendants) to correct outcomes.

We choose G by first computing each child's provisional maximum descendant score,

$$\widetilde{R}(s_{j,t}) \triangleq \max_{k \in m(s_{j,t})} \Big(\gamma^{T_k - t} R_{\text{out}}(q, \tau_k) + R_{\text{fm}}(s_{j,t})\Big). \tag{5}$$

At a fork, we use $G = \max$ if these provisional scores differ across children, and $G = \text{avg}$ otherwise. The averaging case distinguishes children with equal best-case scores by their average descendant outcomes and discounted completion lengths. Appendix A.4 provides an example.

In addition, we bound the formatting reward R_{fm} so that it cannot overturn a nonzero discounted outcome reward under the main training settings. This allows execution feedback to guide learning while preserving the sign of each nonzero discounted outcome term before aggregation. The full definition of R_{fm} is provided in Appendix A.3.

Step 3: Policy Optimization. In the final stage, we compute token-level advantages to guide policy optimization. We compute advantages at two granularities: (i) a *trajectory-relative advantage* $A_{\text{trj}}(\tau_j)$, a coarse-grained signal that scores an entire trajectory among the n candidates using only the outcome reward; and (ii) a *fork-relative advantage* $A_{\text{fork}}(s_{j,t})$, a fine-grained signal that compares a step against its sibling tool-call candidates at the same fork in the rollout tree. Mathematically, denote $\mathsf{norm}(\cdot, \cdot)$ as a z-score normalization function applied to a value with respect to a set of values, and these two advantages are formulated as

$$A_{\text{trj}}(\tau_j) = \mathsf{norm}\big(R_{out}(q, \tau_j), \{R_{out}(q, \tau_k)\}_{k \in [n]}\big), \quad A_{\text{fork}}(s_{j,t}) = \mathsf{norm}\big(R(s_{j,t}), \{R(s_{k,t})\}_{k \in \mathcal{C}(s_{j,t-1})}\big)$$

Notably, singleton and zero-variance groups are assigned zero normalized advantages. For all tokens within a step $s_{j,t}$, the advantage is thereby defined as

$$A(s_{j,t,o}) \triangleq \frac{\omega_1}{|m(s_{j,t})|} \sum_{k \in m(s_{j,t})} A_{\text{trj}}(\tau_k) + \omega_{2,j,t} A_{\text{fork}}(s_{j,t}) \tag{6}$$

where ω_1 controls the trajectory contribution, and $\omega_{2,j,t}$ rescales the fork contribution for the occurrence of step $s_{j,t}$ in trajectory τ_j. Shared steps use the same importance estimate, fork-relative advantage, and averaged trajectory-relative advantage. Their final mixed advantages can differ across occurrences because the rescaling depends on trajectory length. The trajectory-relative advantage provides a coarse-grained measure of a tool-call trajectory's overall effectiveness, offering advantage estimates for every generated token. In contrast, the fork-relative advantage offers a fine-grained signal for distinguishing the best choice at a fork with several competing candidates. We optimize the policy model π_θ in Problem 1 by applying the formulated advantage (i.e., Equation (6)) so that it gradually learns to generate a meaningful tool-call step based on the input context (including a user query and historical tool responses) through trial and error.

Discussion: Setting ω_1 and $\omega_{2,j,t}$. The two terms in Equation (6) use different empirical normalizations: the trajectory term is averaged over trajectories and tokens, while the fork term is naturally averaged over fork nodes, sibling branches, and branch tokens. We therefore rescale the fork term before mixing the two advantages. Let $\mathcal{F}(q) \triangleq \{u : u \text{ is a rollout-tree node and } |\mathcal{C}(u)| > 1\}$ be the set of distinct fork nodes in the rollout tree for query q, and let $n_{\text{forks}}(q) = |\mathcal{F}(q)|$.

Theorem 3.2 (Informal: normalizer-matched rescaling)**.** *For each step $s_{j,t}$ whose parent is a fork, setting $\omega_1 = 1$ and $\omega_{2,j,t} = n|\tau_j| \cdot \big(|m(s_{j,t})||s_{j,t}||\mathcal{C}(s_{j,t-1})|n_{forks}(q)\big)^{-1}$ under the trajectory-wise token averaging in Problem 1, gives a normalizer-matched approximation to the sum of the trajectory-relative and fork-relative objectives, i.e., $J(\theta) \approx J_{trj}(\theta) + J_{fork}(\theta)$.*

A formal analysis is provided in Appendix A.5. This rescaling allows PORTool to use a single token-level advantage in the standard GRPO update while preserving the intended scale of fork-wise comparisons. In summary, PORTool replaces independent rollouts with a rewarded rollout tree and uses Equations (4) and (6) to construct importance-aware advantages, enabling fine-grained credit assignment while remaining trajectory-supervised to address credit-assignment ambiguity.

4 Experiments

4.1 Setup

Dataset. We train and evaluate on an in-the-wild dataset of 3,607 real user queries collected from interactions with a deployed virtual assistant.[1] The queries span **eight** common assistant domains: *Weather, Sports, Geo, Knowledge, Clock, News, Flight,* and *Stock*. Crucially, the dataset contains both (i) time-/location-invariant queries whose answers should remain stable (e.g., general factual questions), and (ii) time-/location-dependent queries whose correct answers evolve with *current time and/or user location* (e.g., "today", "right now", "near me", "latest"). We split the dataset into 3,072 training queries and 535 evaluation queries. Additional details are provided in Appendix C.1.

Tool environment. The tool-use agent interacts with an executable environment consisting of **21 tools** (including the terminal tool `response_gen`). These tools correspond to production-style mobile assistant capabilities, such as weather, news, maps/geo, flights, sports, and finance, as well as utility operators for date/time manipulation, unit conversion, math, and list filtering or sorting. Detailed tool descriptions are provided in Appendix E.

Models and training protocol. We instantiate the tool-use policy with two backbones: *Qwen-2.5-7B-Instruct* [55] and *Qwen-3-1.7B* [56]. We warm-start the agent with supervised fine-tuning and then apply agentic RL with PORTool and baseline algorithms. During RL, each query samples $n = 8$ tool-call trajectories, and each trajectory is capped at six tool-call steps. Unless otherwise specified, PORTool uses a branching factor $f = 4$. Both models are trained for 15 epochs. Full hyperparameters and implementation details are supplemented in Appendix C.

Baselines. We compare PORTool with a diverse set of agentic RL baselines: **GRPO** [42], **DAPO** [60], **ARPO** [5], **AgentFlow** [24], **SimpleTIR** [54], and **ToRL** [23]. These methods cover representative choices in rollout generation, reward design, and policy optimization for tool-use agents. We also construct two variants whose trajectory-level feedback is further augmented by formatting/execution reward, i.e., **GRPO$_{\text{fm}}$** and **ARPO$_{\text{fm}}$**. For fair comparison, all baselines are trained on the same dataset and from the same SFT warm start, rather than using released checkpoints. We follow the original settings of each method, where applicable, while keeping shared hyperparameters, such as the learning rate and the number of rollouts, fixed as listed in Table 4.

Evaluation Metrics and Evaluation Benchmark. For the in-domain results, we conduct five evaluation rounds using the same trained checkpoint for each method. Each round generates one new trajectory for each evaluation query and applies the evaluation protocol stated in Appendix C.3. We compute four metrics within each round. First, **final-answer accuracy** measures whether the generated answer is correct, as assessed by the evaluation agent[2]. Second, **average tool-call steps** measures the number of steps taken before termination, capped at six. Third, **no-final-answer (NFA) rate** measures the fraction of trajectories that do not invoke `response_gen` within the step budget. Fourth, **raw formatting reward** $r_{\text{fm}}(\cdot) \in [0, 1]$ measures formatting and execution quality, averaged over steps within each trajectory and then over queries. We report accuracy as mean $\pm$ standard deviation and the other metrics as means across the five rounds. Policy optimization uses the rescaled reward R_{fm}, defined in Appendix A.3. To evaluate the cross-benchmark generalization, we report results on three benchmarks: **API-Bank** [22], **ToolBench** [36], and **TravelPlanner** [53]. These benchmarks are out of distribution with respect to the training scenario. For ToolBench, we use the StableToolBench [14] setting, in which RapidAPI calls are supported via MirrorAPI. For each benchmark, we follow its official evaluation protocol and report the required metrics.

4.2 Comparison with Other Baselines

Results with Qwen-2.5-7B-Instruct. PORTool achieves the strongest overall performance across accuracy, completion, and efficiency (Table 1). In particular, PORTool reaches 51.96% accuracy,

[1] We only use anonymized text queries and remove any user-identifying information.

[2] The evaluation agent is powered by GPT-4o. On a validation set of around 700 query-answer pairs with human labels, its judgments agree with human annotations in 94.37% of cases.

Table 1: Performance comparisons among different RL training methods on the evaluation dataset

Method	Qwen-2.5-7B-Instruct				Qwen-3-1.7B			
	Accuracy (%)	# Tool-call Steps	NFA Rate (%)	Formatting Reward	Accuracy (%)	# Tool-call Steps	NFA Rate (%)	Formatting Reward
Prompting	24.37 ± 0.75	4.76	58.50	0.466	12.07 ± 1.03	5.50	83.93	0.322
GRPO	46.09 ± 1.22	3.31	13.27	0.808	39.07 ± 0.64	2.96	10.65	0.834
GRPO_{fm}	47.59 ± 1.43	3.18	11.51	0.826	42.77 ± 1.78	2.79	9.91	0.857
DAPO	43.51 ± 1.06	3.37	14.58	0.790	39.48 ± 1.20	3.08	11.96	0.793
ARPO	48.19 ± 0.62	3.19	12.34	0.819	39.25 ± 1.14	3.07	11.96	0.784
ARPO_{fm}	44.97 ± 0.97	3.16	11.55	0.827	34.54 ± 1.05	3.21	11.03	0.808
AgentFlow	48.22 ± 1.06	3.30	11.21	0.816	37.20 ± 1.05	3.21	13.27	0.767
SimpleTIR	47.63 ± 1.16	3.12	10.47	0.832	41.76 ± 1.44	2.82	11.03	**0.884**
ToRL	45.53 ± 1.14	3.20	12.34	0.814	41.72 ± 1.18	2.94	11.40	0.810
PORTool	**51.96** ± 1.09	**3.07**	**7.10**	**0.851**	**46.58** ± 1.22	**2.64**	**7.29**	0.877

Table 2: Performance on three out-of-distribution tasks with Qwen-2.5-7B-Instruct.

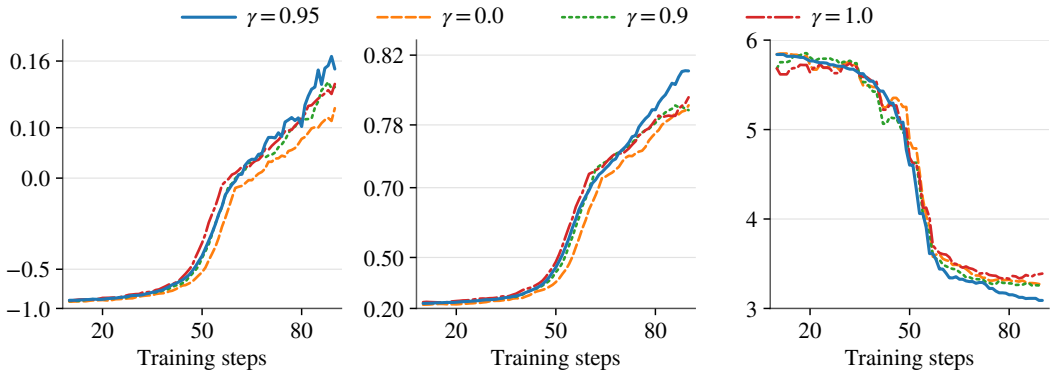
Method	**API-bank Match Rate (%)**				**ToolBench Pass Rate (%)**				**TravelPlanner (%)**	
	Level 1	**Level 2**	**Level 3**	**Avg.**	**I1**	**I2**	**I3**	**Avg.**	**Delivery**	**Pass Avg.**
GRPO_{fm}	65.4	40.3	41.2	57.3	54.4	42.7	48.4	50.4	90.6	16.4
ARPO_{fm}	65.2	38.8	43.5	57.5	54.8	**49.2**	43.4	52.2	92.2	**16.7**
AgentFlow	64.7	41.8	44.3	57.6	50.8	45.1	49.5	49.0	91.7	16.2
SimpleTIR	64.9	**43.3**	42.8	57.6	52.1	48.5	42.1	50.2	88.9	16.2
ToRL	63.9	35.8	39.7	55.4	52.1	45.4	49.0	49.9	86.7	14.8
PORTool	**66.2**	41.8	**45.8**	**59.0**	**56.4**	47.7	**52.2**	**53.4**	**92.8**	**16.7**

exceeding AgentFlow by 3.74 percentage points. At the same time, it reduces the average number of tool-call steps to 3.07 and lowers the no-final-answer rate to 7.10%. These results indicate that PORTool improves the quality of tool-use decisions without relying on longer trajectories.

We attribute this improvement to PORTool's importance-aware credit assignment. Most trajectory-supervised baselines (e.g., GRPO and DAPO) broadcast outcome supervision uniformly across steps, which obscures which tool-use decision at a branching point caused success or failure. Prefix-sharing methods such as ARPO and AgentFlow improve sampling efficiency but still rely mainly on trajectory-level updates. In contrast, PORTool introduces fork-relative advantages that compare sibling actions under the same prefix. This allows the policy to reinforce better tool-use decisions while avoiding unnecessary penalties for shared-prefix steps. Augmenting trajectory rewards with formatting feedback (e.g., GRPO_{fm}) improves executability, but they lack a tool-use decision comparison mechanism, which explains why PORTool achieves higher correctness with fewer tool calls.

Results with Qwen-3-1.7B. The same trend holds for the smaller Qwen-3-1.7B backbone. Agentic RL substantially reduces formatting and execution failures compared with prompting. Among agentic RL methods, PORTool again achieves the best final-answer accuracy and the lowest no-final-answer rate, while also requiring the fewest tool-call steps. Compared with the highest-accuracy baseline, GRPO_{fm}, PORTool improves accuracy by 3.81 percentage points, reduces the no-final-answer rate by 2.62 percentage points, and shortens trajectories. Although SimpleTIR achieves the best formatting reward, PORTool remains very close while substantially improving correctness, indicating that its gains come not merely from stricter formatting but from better tool-use decisions and more reliable recovery from tool feedback.

Cross-benchmark Generalization. Table 2 reports results on API-Bank, ToolBench, and TravelPlanner, where the tool inventories and task requirements differ from those used during training. Admittedly, the superiority over baseline methods on these out-of-distribution tasks is more modest than on the in-domain evaluation set. This is expected because the tools and prompts differ substantially from the training environment. Nevertheless, PORTool achieves the best average match rate on API-Bank, the best average pass rate on ToolBench, and the highest delivery pass rate on TravelPlanner, while tying the best average TravelPlanner score. These results suggest that PORTool learns a transferable tool-use policy rather than memorizing benchmark-specific tool routines. Its



(a) Outcome reward (b) Formatting reward (c) Tool-call steps

Figure 3: Comparison of different decay factors γ of Equation (4).

learning signal uses outcome supervision and generic execution validity, then converts them into prefix-local, sibling-normalized comparisons. This design remains effective across different tool sets, task requirements, and interaction patterns.

4.3 Discussions

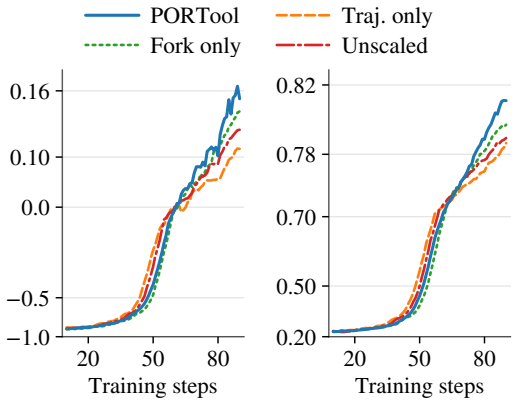
Effect of the Decay Factor γ. The decay factor γ controls the step-wise importance score in Equation (4). Figure 3 compares four values of γ and tracks outcome reward, formatting reward, and tool-call steps during training. A clear pattern is that $\gamma = 0.95$ offers the best overall balance among correctness, formatting robustness, and efficiency. It achieves the highest final outcome reward and formatting reward while also converging to the fewest tool-call steps.

When γ is too small (e.g., $\gamma = 0.0$), the discounted outcome term vanishes for non-terminal steps, so step-wise importance is dominated by the formatting/execution signal, and the agent receives weak guidance for long-horizon correctness. This leads to slower improvement and lower final outcome reward. In contrast, $\gamma = 1.0$ propagates outcome reward equally to all preceding steps, regardless of the remaining horizon. This removes the length-sensitive preference and can encourage unnecessarily long tool interactions. Consistent with Figure 3c, $\gamma = 1.0$ tends to converge to more tool-call steps than $\gamma = 0.95$, even when the final outcome reward becomes competitive. Based on these observations, we adopt $\gamma = 0.95$ in all main experiments.

Effect of the Advantage Coefficients. Figure 4 studies how different ways of combining trajectory-relative and fork-relative advantages affect optimization. We compare four variants: (i) PORTool using the scaled coefficients implied by Theorem 3.2, (ii) **traj.-only** with $\omega_{2,j,t} = 0$, (iii) **fork-only** with $\omega_1 = 0$, and (iv) **unscaled** with $\omega_1 = 1$ and $\omega_{2,j,t} = 1$.

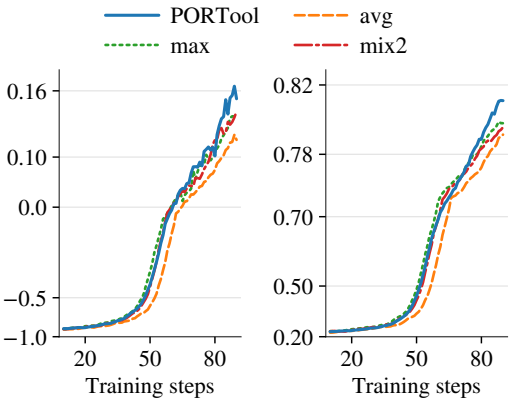
The results show that both advantage components are necessary, and that proper scaling matters. The trajectory-only variant performs poorly because it assigns the same trajectory-level signal to all steps, without distinguishing competing tool choices at a fork. The fork-only variant performs much better because it directly compares sibling actions under the shared prefix. However, it still underperforms PORTool since fork-relative signals are only available at forked steps and become zero when no meaningful sibling comparison exists. This leaves many tokens without informative credit. The unscaled setting improves over trajectory-only but consistently underperforms PORTool, showing that naively mixing two differently normalized advantages can distort credit assignment. These results empirically support the theoretically motivated scaling in Theorem 3.2.

Effect of Reward Function $R(s_{j,t})$ Design. Figure 5 evaluates alternative designs of the aggregation operator $G(\cdot)$ in Equation (4), which computes step-wise importance from descendant outcomes. We compare PORTool's adaptive aggregation rule with three variants: (i) **avg**: always averaging descendant values; (ii) **max**: always taking the maximum; and (iii) **mix2**: a mixed variant that aggregates only the discounted outcome term while adding the formatting/execution reward outside the aggregation.



(a) Outcome reward (b) Formatting reward

Figure 4: Different advantage settings



(a) Outcome reward (b) Formatting reward

Figure 5: Different designs of $G(\cdot)$ in Eq. (4)

PORTool achieves the best final outcome and formatting rewards. The always-average variant performs worst because averaging can dilute rare but important successful branches at a fork, weakening the relative signal needed to learn decisive tool choices. The always-max variant is stronger than always-average, but it can over-emphasize a single high-reward descendant (including lucky rollouts). This increases variance in step credit assignment and slightly degrades the final plateau compared to the adaptive rule. Finally, mix2 underperforms PORTool because separating formatting/execution reward from the aggregation reduces the role of execution quality when deciding whether sibling branches should be treated as tied. Empirically, this leads to lower formatting reward and slightly weaker outcome reward than PORTool. These comparisons support the adaptive $G(\cdot)$ design as a practical way to preserve strong learning signals at forks while avoiding brittle, high-variance updates.

5 Conclusion

In this work, we study trajectory-supervised agentic RL for enabling multi-tool-integrated reasoning under executable, dynamic tool feedback. We identify *credit-assignment ambiguity* as a key limitation of optimization based solely on outcome rewards. Assigning the same trajectory-level reward to every step obscures which tool decisions contribute to success. It can even suppress steps necessary for correctness. To address this, we propose PORTool, which organizes rollouts into a *rewarded, prefix-sharing rollout tree* to create *prefix-matched within-context comparisons* at tool-call forks, and learns *prefix-consistent* step-wise importance from terminal correctness augmented with bounded execution/format feedback. Using a mixed advantage that combines coarse trajectory-relative and fine-grained fork-relative advantage, PORTool consistently improves final-answer accuracy while reducing tool-call steps and failure modes compared with state-of-the-art policy-optimization baselines.

Limitations

Our main experiments focus on 7B-scale and smaller models, which suit efficient deployment and better match the intended use case of on-device assistant queries. However, we have not yet fully evaluated whether the same gains hold for much larger models, stronger proprietary models, or models from a broader range of families. We use the Qwen model family because it is widely adopted in recent LLM research and offers strong open-weight backbones for tool-use training, but future work should test PORTool across more model architectures and scales.

Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive comments. We gratefully acknowledge support from Apple and the National Science Foundation. This work was supported in part by the National Science Foundation under grant NSF-IIS 2141037. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors declare no additional competing interests beyond the affiliations and funding disclosed above.

References

[1] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267. Association for Computational Linguistics, 2024.

[2] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[3] Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 10818–10838. PMLR, 2025.

[4] Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. GPG: A simple and strong reinforcement learning baseline for model reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026.

[5] Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic reinforced policy optimization. In *The Fourteenth International Conference on Learning Representations*, pages 16981–17017, 2026.

[6] Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Model alignment as prospect theoretic optimization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 12634–12651. PMLR, 2024.

[7] Wenzhi Fang, Dong-Jun Han, Liangqi Yuan, Evan Chen, and Christopher Brinton. Bridging on-device and cloud LLMs for collaborative reasoning: A unified methodology for local routing and post-training. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 29029–29059. PMLR, 2026.

[8] Wenzhi Fang, Liangqi Yuan, Guangchen Lan, Dong-Jun Han, and Christopher G. Brinton. Iterative critique-and-routing controller for multi-agent systems with heterogeneous LLMs. In *Advances in Neural Information Processing Systems*, 2026. To appear.

[9] Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. ReTool: Reinforcement learning for strategic tool use in LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026.

[10] Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems*, volume 38, pages 46375–46408, 2025.

[11] Dawei Gao, Zitao Li, Yuexiang Xie, Weirui Kuang, Liuyi Yao, Bingchen Qian, Zhijian Ma, Yue Cui, Haohao Luo, Shen Li, Lu Yi, Yi Yu, Shiqi He, Zhiling Luo, Wenmeng Zhou, Zhicheng Zhang, Xuguang He, Ziqian Chen, Weikai Liao, Farruh Isakulovich Kushnazarov, Yaliang Li, Bolin Ding, and Jingren Zhou. AgentScope 1.0: A developer-centric framework for building agentic applications. *arXiv preprint arXiv:2508.16279*, 2025.

[12] Xingjie Gao, Pengcheng Huang, Zhenghao Liu, Yukun Yan, Shuo Wang, Zulong Chen, Chen Qian, Ge Yu, and Yu Gu. Teaching LLMs to learn tool trialing and execution through environment interaction. *arXiv preprint arXiv:2601.12762*, 2026.

[13] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[14] Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156. Association for Computational Linguistics, 2024.

[15] Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning. *arXiv preprint arXiv:2403.04642*, 2024.

[16] Shiqi He, Yue Cui, Xinyu Ma, Yaliang Li, Bolin Ding, and Mosharaf Chowdhury. Branch-and-browse: Efficient and controllable web exploration with tree-structured reasoning and action memory. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18407–18418. Association for Computational Linguistics, 2026.

[17] Shiqi He, Yue Cui, Feijie Wu, Xinyu Ma, Jiaheng Lu, Yaliang Li, Bolin Ding, and Mosharaf Chowdhury. Beyond domains: Reusing web skills via transferable interaction patterns. In *Advances in Neural Information Processing Systems*, 2026. To appear.

[18] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[19] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025.

[20] Doyoung Kim, Zhiwei Ren, Jie Hao, Zhongkai Sun, Lichao Wang, Xiyao Ma, Xu Han, Jun Yin, Heng Ji, Wei Shen, Xing Fan, Benjamin Z. Yao, and Chenlei Guo. Beyond perfect APIs: Evaluating LLM agents under realistic API complexity. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, pages 44744–44774. Association for Computational Linguistics, 2026.

[21] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626. ACM, 2023.

[22] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116. Association for Computational Linguistics, 2023.

[23] Xuefeng Li, Haoyang Zou, and Pengfei Liu. ToRL: Scaling tool-integrated RL. *arXiv preprint arXiv:2503.23383*, 2025.

[24] Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use. In *The Fourteenth International Conference on Learning Representations*, 2026.

[25] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024.

[26] Heng Lin and Zhongwen Xu. Understanding tool-integrated reasoning. In *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS*, 2025.

[27] Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. ToolACE:

Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations*, pages 41359–41381, 2025.

[28] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-like training: A critical perspective. In *Second Conference on Language Modeling*, 2025.

[29] Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. In *Advances in Neural Information Processing Systems*, volume 37, pages 54463–54482, 2024.

[30] Yifei Lu, Fanghua Ye, Jian Li, Qiang Gao, Cheng Liu, Haibo Luo, Nan Du, Xiaolong Li, and Feiliang Ren. CodeTool: Enhancing programmatic tool invocation of LLMs via process supervision. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18287–18304. Association for Computational Linguistics, 2025.

[31] Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhu Chen. General-Reasoner: Advancing LLM reasoning across all domains. In *Advances in Neural Information Processing Systems*, volume 38, pages 56596–56618, 2025.

[32] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[33] Aaron Parisi, Yao Zhao, and Noah Fiedel. TALM: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

[34] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565, 2024.

[35] Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. ToolRL: Reward is all tool learning needs. In *Advances in Neural Information Processing Systems*, volume 38, pages 105523–105553, 2025.

[36] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024.

[37] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025.

[38] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741, 2023.

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

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

[41] Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025.

[42] 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. *arXiv preprint arXiv:2402.03300*, 2024.

[43] 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. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297. ACM, 2025.

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

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

[46] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

[47] Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, Weizhu Chen, Shuohang Wang, Simon S. Du, and Yelong Shen. Reinforcement learning for reasoning in large language models with one training example. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

[48] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837, 2022.

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

[50] Feijie Wu, Zitao Li, Fei Wei, Yaliang Li, Bolin Ding, and Jing Gao. Talk to right specialists: Iterative routing in multi-agent systems for question answering. *arXiv preprint arXiv:2501.07813*, 2025.

[51] Feijie Wu, Xiaoze Liu, Haoyu Wang, Xingchen Wang, Lu Su, and Jing Gao. Towards federated RLHF with aggregated client preference for LLMs. In *The Thirteenth International Conference on Learning Representations*, 2025.

[52] Feijie Wu, Hugo Barbalho, Konstantina Mellou, Marco Molinaro, Jing Gao, Ishai Menache, Xinzhi Zhang, and Sirui Li. HeurEvo: Agentic evolution of hybrid solver-augmented heuristics for time-critical mathematical optimization. *arXiv preprint arXiv:2609.36303*, 2026.

[53] Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A benchmark for real-world planning with language agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 54590–54613. PMLR, 2024.

[54] Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. SimpleTIR: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. In *The Fourteenth International Conference on Learning Representations*, pages 8424–8449, 2026.

[55] An Yang, Bao-Song Yang, Bei-Chen Zhang, Bin-Yuan Hui, Bo Zheng, Bo-Wen Yu, Cheng-Yuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Hao-Ran Wei, Huan Lin, Jian Yang, Jian-Hong Tu, Jian-Wei Zhang, Jian-Xin Yang, Jia-Xin Yang, Jing-Ren Zhou, Jun-Yang Lin, Kai Dang,

Ke-Ming Lu, Ke-Qin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Run-Ji Lin, Tian-Hao Li, Tingyu Xia, Xing-Zhang Ren, Xuan-Cheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhen-Ru Zhang, Zi-Han Qiu, Shanghaoran Quan, and Zekun Wang. Qwen2.5 technical report. *ArXiv*, abs/2412.15115, 2024.

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

[57] Shiming Yang, Yuxuan Tong, Xinyao Niu, Graham Neubig, and Xiang Yue. Demystifying long chain-of-thought reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 71177–71209. PMLR, 2025.

[58] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822, 2023.

[59] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[60] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

[61] Yuanqing Yu, Zhefan Wang, Weizhi Ma, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. StepTool: Enhancing multi-step tool usage in LLMs via step-grained reinforcement learning. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pages 3952–3962. ACM, 2025.

[62] Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, and Mingyi Hong. Reinforcing multi-turn reasoning in LLM agents via turn-level credit assignment. *arXiv preprint arXiv:2505.11821v1*, 2025.

[63] Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. SimpleRL-Zoo: Investigating and taming zero reinforcement learning for open base models in the wild. In *Second Conference on Language Modeling*, 2025.

[64] Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhong-Zhi Li, Xiangyuan Xue, Yijiang Li, Yifan Zhou, Yang Chen, Chen Zhang, Yutao Fan, Zihu Wang, Songtao Huang, Francisco Piedrahita-Velez, Yue Liao, Hongru Wang, Mengyue Yang, Heng Ji, Jun Wang, Shuicheng Yan, Philip Torr, and Lei Bai. The landscape of agentic reinforcement learning for LLMs: A survey. *Transactions on Machine Learning Research*, 2026.

[65] Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. Nemotron-Research-Tool-N1: Exploring tool-using language models with reinforced reasoning. In *The Fourteenth International Conference on Learning Representations*, pages 91437–91453, 2026.

[66] Yifan Zhang and Math-AI Team. American invitational mathematics examination (AIME) 2025, 2025. Dataset.

[67] Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

[68] Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. The surprising effectiveness of negative reinforcement in LLM reasoning. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Algorithm 1 Tree Rollout

Input: Query q, number of rollouts n, branching factor f, maximum depth $T_{\max}$, tool-use LLM π_θ, tool set U
Required: IsCompleted(τ) returns whether τ is completed (i.e., the last step should include the tool call `response_gen`).
Required: DupAndSamp(`traj`, f, N) duplicates each element in `traj` f times, then samples N elements and returns them.
`completed` $\leftarrow [\,]$
▷ *Initialize n trajectories with the first tool-call step*
`traj` $\leftarrow \emptyset$
for $j = 1, \ldots, n$ **do**
  $s_{j,1} \sim \pi_\theta(s \mid q)$
  $a_{j,1} \leftarrow U(s_{j,1})$
  $\tau_j \leftarrow [\,q,\ (s_{j,1}, a_{j,1})\,]$
  `traj` $\leftarrow$ `traj` $\cup \{\tau_j\}$
end for
▷ *Iterative expansion with branching factor f while keeping exactly n trajectories*
for $t = 2, \ldots, T_{\max}$ **do**
  for all $\tau \in$ `traj` **do**
    if IsCompleted(τ) **then**
      `completed.append`(τ)
      `traj` $\leftarrow$ `traj` $\setminus \{\tau\}$
    end if
  end for
  $n_c \leftarrow |$`completed`$|$
  if $n_c = n$ **then**
    break
  end if
  `traj` $\leftarrow$ DupAndSamp(`traj`, $f, n - n_c$)
  for all $\tau \in$ `traj` **do**
    $s_t \sim \pi_\theta(s \mid q, \tau)$
    $a_t \leftarrow U(s_t)$
    τ`.append`$((s_t, a_t))$
  end for
end for
`completed.extend`(`traj`)
return `completed`

A Additional Details of PORTool

A.1 Tree Rollout

Tree-structured exploration enables effective comparisons of alternative decisions under shared contexts [16]. For each query q, PORTool initializes n first-step candidates independently. Subsequent expansions use branching factor f, while completed trajectories are retained. The resulting rollouts naturally induce a tree structure: trajectories share prefixes (common histories) and diverge only at selected steps. Algorithm 1 provides the official pseudocode of the rollout procedure used in this work.

A.2 Rollout-Tree Notations

A completed rollout for query q is a tool-call trajectory $\tau_j = \{q, (s_{j,1}, a_{j,1}), \ldots, (s_{j,T_j}, a_{j,T_j})\}$. At each step, $s_{j,t}$ is the tool-use agent's generated content (reasoning traces plus one or more structured tool calls), and $a_{j,t} = U(s_{j,t})$ is the corresponding tool response (or error message).

Shared-step index set $m(\cdot)$. We represent the n rollouts as a tree where each node corresponds to a *shared prefix step* (i.e., a step reached under the same interaction history). For any node/step $s_{j,t}$,

define

$$m(s_{j,t}) \triangleq \{k \in [n] : \text{trajectory } \tau_k \text{ passes through the same node as } s_{j,t}\}.$$

Intuitively, $m(s_{j,t})$ contains all trajectory indices that share the same prefix up to (and including) step t.

Child set $C(\cdot)$. For any step (node) $s_{j,t}$, define its child set $C(s_{j,t})$ as the set of trajectory indices corresponding to *distinct* $(t+1)$-th steps branching from $s_{j,t}$. By construction, for any $k_1 \neq k_2 \in C(s_{j,t})$, we have $s_{k_1,t+1} \neq s_{k_2,t+1}$. It is possible that two distinct trajectories $k_1, k_2 \in m(s_{j,t})$ generate the same next step (duplicate children), so $C(s_{j,t}) \subseteq m(s_{j,t})$.

Fork set and fork count. Let s_0 be a virtual root representing the query and initial context, with $s_{j,0} = s_0$ for every trajectory. Define the set of distinct rollout-tree nodes as

$$V(q) = \{s_0\} \cup \{s_{j,t} : j \in [n],\ t \in [T_j]\}.$$

A node is a fork if it has more than one distinct child:

$$F(q) = \{u \in V(q) : |C(u)| > 1\}, \qquad n_{\text{forks}}(q) = |F(q)|.$$

A.3 Reward Components and Rescaled Formatting Reward

PORTool computes step-wise importance by combining (i) the evaluation agent's trajectory outcome reward and (ii) an auxiliary formatting/execution reward.

Outcome reward R_{out}. After a rollout terminates, the evaluation agent assigns

$$R_{\text{out}}(q, \tau_j) \in \{+1, 0, -1\},$$

corresponding to {true, unable_to_answer, false}.

Raw formatting score $r_{\text{fm}} \in [0,1]$. For each step $s_{j,t}$, we compute a raw formatting score using the following rubric:

1. **Reasoning block:** if the output includes a `<think> ... </think>` block, add $+0.2$.
2. **Formatting checks (activated only if the reasoning block is present):**
   - If the tool-call wrapper `<tool_call> ... </tool_call>` is included, add $+0.1$.
   - If the enclosed content can be parsed as valid JSON, add $+0.1$.
   - If every tool call includes the required fields `"name"` (string) and `"arguments"` (dictionary), add $+0.05$.
   - **Tool invocation success:** if all tool calls succeed, add $+0.55$; otherwise assign partial credit proportional to the fraction of successful tool calls.

By construction, $r_{\text{fm}}(s_{j,t}) \in [0,1]$.

Bounded formatting/execution reward. To ensure that nonzero discounted outcome rewards dominate the formatting term in magnitude, we rescale the raw formatting score $r_{\text{fm}}(\cdot)$ to a bounded, centered reward:

$$R_{\text{fm}}(s_{j,t}) \triangleq \alpha\Big(r_{\text{fm}}(s_{j,t}) - \tfrac{1}{2}\Big), \qquad R_{\text{fm}}(s_{j,t}) \in \big[-\tfrac{\alpha}{2}, \tfrac{\alpha}{2}\big].$$

For $\gamma \in (0,1]$, let $H_{\max} = \max_{j,t}(T_j - t)$. Choosing $\alpha < \gamma^{H_{\max}}$ preserves the ordering of individual descendant values with different outcome labels before aggregation. With six tool-call steps, $H_{\max} = 5$. We use $\gamma = 0.95$ and $\alpha = 0.5$, yielding $R_{\text{fm}} \in [-0.25, 0.25]$. After aggregation, the step score combines descendant outcomes with this bounded auxiliary term. This dominance condition does not apply to the $\gamma = 0$ ablation.

A.4 Step-wise Reward and Adaptive Aggregation G

Given a step (node) $s_{j,t}$, we define its step-wise reward by aggregating descendant outcomes plus the formatting reward:

$$R(s_{j,t}) \triangleq G\Big(\big\{\gamma^{T_k - t} R_{\text{out}}(q, \tau_k) + R_{\text{fm}}(s_{j,t})\big\}_{k \in m(s_{j,t})}\Big), \tag{7}$$

where $\gamma \in [0,1]$ is the decay factor and $G(\cdot)$ is chosen adaptively to stabilize credit assignment at forks.

Adaptive rule for G. Let p be the parent of $s_{j,t}$ (i.e., $p = s_{j,t-1}$) and let $\mathrm{Ch}(p)$ denote the set of distinct child steps under p. Define the set of *provisional* values

$$\widetilde{R}(s) \triangleq \max_{k \in m(s)} \Big(\gamma^{T_k - t} R_{\mathrm{out}}(q, \tau_k) + R_{\mathrm{fm}}(s) \Big), \qquad s \in \mathrm{Ch}(p).$$

If the children are discriminative (i.e., $\{\widetilde{R}(s) : s \in \mathrm{Ch}(p)\}$ is not a singleton), we set $G = \max$ in equation 7 to emphasize the best branch. Otherwise (children are effectively tied), we set $G = \mathrm{avg}$ so that the score reflects reliability rather than a single lucky rollout.

Toy example: when $G = \mathrm{avg}$ (tied best-case children). Recall that for a node/step s we aggregate the set of descendant values $v_k(s) \triangleq \gamma^{T_k - t} R_{\mathrm{out}}(q, \tau_k) + R_{\mathrm{fm}}(s)$ over $k \in m(s)$, and define $R(s) = G(\{v_k(s)\}_{k \in m(s)})$. At a fork parent step p with multiple children $s \in \mathrm{Ch}(p)$, we use the adaptive rule: we first compute each child's *best-case* provisional value $R^{\max}(s) = \max_{k \in m(s)} v_k(s)$; if $\{R^{\max}(s)\}_{s \in \mathrm{Ch}(p)}$ is not a singleton, we set $G = \max$; otherwise (children are best-case tied) we set $G = \mathrm{avg}$.

Setup. Consider a fork with two candidate tool-call steps s^A and s^B under the same prefix history. Assume $R_{\mathrm{fm}}(s^A) = R_{\mathrm{fm}}(s^B) = 0$ and $\gamma = 1$ for simplicity, so $v_k(s) = R_{\mathrm{out}}(q, \tau_k) \in \{+1, -1\}$. Suppose we sample four continuations under each child:

Child step	Outcomes of its descendant rollouts	$R^{\max}(s)$
s^A	$\{+1, -1, -1, -1\}$	$+1$
s^B	$\{+1, +1, +1, -1\}$	$+1$

Why $\max$ is insufficient here. Both children have the same best-case value ($R^{\max}(s^A) = R^{\max}(s^B) = +1$), meaning that *each child has at least one successful descendant*. If we used $G = \max$ unconditionally, then $R(s^A) = R(s^B) = +1$ and the fork-level comparison cannot prefer the more reliable tool decision.

Why the adaptive rule chooses avg. Because the best-case values are tied, we set $G = \mathrm{avg}$ and obtain

$$R(s^A) = \mathrm{avg}(\{+1, -1, -1, -1\}) = -0.5, \qquad R(s^B) = \mathrm{avg}(\{+1, +1, +1, -1\}) = +0.5.$$

Thus, although both children can sometimes lead to success, averaging assigns higher importance to s^B because it succeeds more consistently. In this way, the avg case measures *reliability* (and, when $\gamma < 1$, also *average efficiency* via discounting), rather than being dominated by a single lucky successful rollout.

Contrast: when the rule uses $\max$. If instead s^A *never* leads to a correct completion (e.g., $\{-1, -1, -1, -1\}$), then $R^{\max}(s^A) = -1$ and $R^{\max}(s^B) = +1$ are discriminative; the adaptive rule sets $G = \max$ to emphasize that s^B is the only branch that can reach a correct outcome.

A.5 Fork-Relative Advantages and Formal Approximation of Theorem 3.2

Theorem 3.2 provides an overview of rescaling the fork coefficient. In the following theorem, we give a formal statement showing why the mixed objective approximately equals the sum of the trajectory-relative and fork-relative objectives, together with an error bound for this approximation.

Theorem A.1 (Formal approximation of normalizer-matched rescaling)**.** *Consider the token-level objective in Problem 1 with the mixed advantage in Equation (6). Set $\omega_1 = 1$. For every step $s_{j,t}$ whose parent $s_{j,t-1}$ is a fork, set*

$$\omega_{2,j,t} = \frac{n|\tau_j|}{|m(s_{j,t})|\,|s_{j,t}|\,|\mathcal{C}(s_{j,t-1})|\,n_{\mathrm{forks}}(q)}.$$

For steps that are not children of a fork, set the fork contribution to zero. Let $J_{\mathrm{trj}}(\theta)$ be the same objective using only the trajectory-relative term in Equation (6), and let $J_{\mathrm{fork}}(\theta)$ be the fork-wise objective that averages over fork nodes, their child branches, and child-branch tokens:

$$J_{\mathrm{fork}}(\theta) = \mathbb{E}\left[\frac{1}{n_{\mathrm{forks}}(q)} \sum_{s_{j,t} \in \mathcal{F}(q)} \frac{1}{|\mathcal{C}(s_{j,t})|} \sum_{k \in \mathcal{C}(s_{j,t})} \frac{1}{|s_{k,t+1}|} \sum_{o=1}^{|s_{k,t+1}|} f_\theta^{(fork)}(s_{k,t+1,o}) \right],$$

where per-token objective on the fork is defined as

$$f_\theta^{(fork)}(s_{j,t,o}) \triangleq \min\left(\rho_\theta(s_{j,t,o})A_{\text{fork}}(s_{j,t}), \text{clip}_{1-\epsilon}^{1+\epsilon}(\rho_\theta(s_{j,t,o}))A_{\text{fork}}(s_{j,t}).\right)$$

If $n_{\text{forks}}(q) = 0$, we define $J_{\text{fork}}(\theta) = 0$. Then the mixed objective satisfies

$$J(\theta) = J_{\text{trj}}(\theta) + J_{\text{fork}}(\theta) + \Delta_{\text{clip}}(\theta),$$

where the clipping-induced residual is

$$\Delta_{\text{clip}}(\theta) = \mathbb{E}\left[\frac{1}{n}\sum_{j=1}^{n}\frac{1}{|\tau_j|}\sum_{t=1}^{T_j}\sum_{o=1}^{|s_{j,t}|}\left|\rho_\theta(s_{j,t,o}) - \text{clip}_{1-\epsilon}^{1+\epsilon}(\rho_\theta(s_{j,t,o}))\right|\min\left(|a|,|b|\right)\mathbf{1}\left\{ab<0\right\}\right],$$

given

$$a = \frac{1}{|m(s_{j,t})|}\sum_{k\in m(s_{j,t})}A_{\text{trj}}(\tau_k), \qquad b = \omega_{2,j,t}A_{\text{fork}}(s_{j,t}),$$

Therefore, the bound for the clipping-induced residual is

$$0 \le \Delta_{\text{clip}}(\theta) \le \mathbb{E}\left[\frac{1}{n}\sum_{j=1}^{n}\frac{1}{|\tau_j|}\sum_{t=1}^{T_j}\sum_{o=1}^{|s_{j,t}|}\left|\rho_\theta(s_{j,t,o}) - \text{clip}_{1-\epsilon}^{1+\epsilon}(\rho_\theta(s_{j,t,o}))\right|\min\left(|a|,|b|\right)\right].$$

When the clipping-induced residual is small, we have $J(\theta) \approx J_{\text{trj}}(\theta) + J_{\text{fork}}(\theta)$.

Proof. We prove the statement for one sampled rollout tree. Taking expectation over queries and rollout trees gives the result for $J(\theta)$.

First, consider the fork-relative part before the clipping interaction is considered. The clipped GRPO term is positively homogeneous in the advantage. For any $c \ge 0$,

$$\min\left(\rho ca, \text{clip}_{1-\epsilon}^{1+\epsilon}(\rho)ca\right) = c\min\left(\rho a, \text{clip}_{1-\epsilon}^{1+\epsilon}(\rho)a\right).$$

Therefore, multiplying $A_{\text{fork}}(s_{j,t})$ by $\omega_{2,j,t}$ is equivalent to multiplying the corresponding fork loss by $\omega_{2,j,t}$.

In Equation (1), every token in trajectory τ_j is averaged with weight

$$\frac{1}{n}\cdot\frac{1}{|\tau_j|}.$$

For a child step $s_{j,t}$ of a forked parent $s_{j,t-1}$, multiplying the fork advantage by the proposed $\omega_{2,j,t}$ changes this weight to

$$\frac{1}{n}\cdot\frac{1}{|\tau_j|}\cdot\frac{n|\tau_j|}{|m(s_{j,t})|\,|s_{j,t}|\,|\mathcal{C}(s_{j,t-1})|\,n_{\text{forks}}(q)} = \frac{1}{|m(s_{j,t})|\,|s_{j,t}|\,|\mathcal{C}(s_{j,t-1})|\,n_{\text{forks}}(q)}.$$

The same child step appears in all trajectories indexed by $m(s_{j,t})$. Summing over these repeated trajectory occurrences cancels the factor $|m(s_{j,t})|$. Hence each token of this child step receives weight

$$\frac{1}{|s_{j,t}|\,|\mathcal{C}(s_{j,t-1})|\,n_{\text{forks}}(q)}.$$

This is exactly the fork-wise averaging weight: first average over fork nodes, then over child branches, and then over tokens in each child branch. Thus the proposed $\omega_{2,j,t}$ matches the normalizer of $J_{\text{fork}}(\theta)$.

It remains to quantify the gap between using one mixed clipped term and using two separately clipped terms. For any two advantage values a and b, a direct case check gives

$$\begin{aligned}&\min\left(\rho(a+b), \text{clip}_{1-\epsilon}^{1+\epsilon}(\rho)(a+b)\right)\\ &\quad= \min\left(\rho a, \text{clip}_{1-\epsilon}^{1+\epsilon}(\rho)a\right) + \min\left(\rho b, \text{clip}_{1-\epsilon}^{1+\epsilon}(\rho)b\right) + \left|\rho - \text{clip}_{1-\epsilon}^{1+\epsilon}(\rho)\right|\min(|a|,|b|)\mathbf{1}\{ab<0\}.\end{aligned} \tag{8}$$

Applying this identity with

$$a = \frac{1}{|m(s_{j,t})|} \sum_{k \in m(s_{j,t})} A_{\mathrm{trj}}(\tau_k), \qquad b = \omega_{2,j,t} A_{\mathrm{fork}}(s_{j,t})$$

gives the stated expression for $\Delta_{\mathrm{clip}}(\theta)$. Since each term in $\Delta_{\mathrm{clip}}(\theta)$ is nonnegative, the lower bound follows. The upper bound follows by removing the sign-disagreement indicator. This proves the theorem. □

Remark. The residual shows that the approximation error is localized. It appears only when two events happen at the same time: the likelihood ratio leaves the clipping interval, and the trajectory-relative and fork-relative advantages disagree in sign.

The residual is zero whenever

$$\rho_\theta(s_{j,t,o}) = \mathrm{clip}_{1-\epsilon}^{1+\epsilon}(\rho_\theta(s_{j,t,o})).$$

This holds for all tokens whose likelihood ratios remain inside the PPO/GRPO clipping interval. The residual is also zero whenever the trajectory-level and fork-level signals have the same sign. In this case, both signals agree on whether the token should be encouraged or discouraged, so the clipped objective remains additive for these two components.

At the beginning of each PPO/GRPO update, the current policy equals the behavior policy, so

$$\rho_\theta(s_{j,t,o}) = 1.$$

Since 1 lies inside the clipping interval, the residual is exactly zero at the start of the update. Therefore, the mixed objective and the separate objective have the same first-order behavior around θ_{old}. During later minibatch updates, the error is controlled by the amount of ratio clipping on tokens where the two advantage components disagree in sign. Thus, the approximation is accurate when updates are proximal and when the two credit signals usually agree in sign.

This rescaled mixed advantage folds the fork-wise objective into the standard token-level GRPO objective. PORTool still uses the rollout tree to compute A_{fork}, but the policy update only needs one scalar advantage for each token and one clipped GRPO loss. This avoids a separate fork-loss pass, extra fork-child gathering during loss computation, and repeated clipped-loss reductions for trajectory and fork terms. As a result, the method is easier to implement in existing GRPO training code and reduces the optimization-stage overhead compared with a naive implementation that separately optimizes trajectory and fork losses, while preserving the intended scale of the fork-wise signal.

B Proof of Theorem 3.1

Assumptions. We analyze one query q and one expected GRPO-style update around θ_{old}.

(A1) **Binary terminal outcome.** Each trajectory τ receives an outcome reward $r(\tau) \in \{+1, -1\}$, where $+1$ means correct and -1 means incorrect.

(A2) **Necessary step.** Let E denote the event that a trajectory contains s^*. Necessity means

$$\Pr_{\theta_{\text{old}}}(\text{correct} \mid \neg E) = 0.$$

Thus every correct trajectory must contain s^*.

(A3) **Z-score trajectory advantage.** GRPO assigns each trajectory a scalar advantage $A(\tau)$ by z-score normalizing $r(\tau)$. For analysis, we use the population form

$$A(\tau) = \frac{r(\tau) - \mu}{\sigma}, \qquad \mu = \mathbb{E}[r], \qquad \sigma = \sqrt{\mathbb{E}[(r-\mu)^2]}.$$

We assume $\sigma > 0$, i.e., both correct and incorrect trajectories exist.

(A4) **Trajectory-level credit assignment with per-token averaging.** The scalar advantage $A(\tau)$ is broadcast to all tokens in τ, and the token-level loss is averaged over trajectory length, giving a weight $1/|\tau|$.

(A5) **Common local decision context.** Every analyzed trajectory visits the same decision context h^* exactly once. We treat s^* as a categorical action at this context and update its local softmax logit z, with all other policy parameters fixed. Let $p_{s^*}(\theta) \in (0, 1)$ denote its selection probability. Then

$$\frac{\partial}{\partial z} \log \pi_\theta(a \mid h^*) = \mathbf{1}\{a = s^*\} - p_{s^*}(\theta).$$

Under this model, selecting s^* is equivalent to the trajectory event E.

(A6) **Expected small-step update.** We analyze an update of the local logit using the expected GRPO-style gradient. The positive step size is sufficiently small that the first-order term determines the sign of the change in $\log p_{s^*}(\theta)$.

Role of the local decision model. Both trajectories selecting s^* and trajectories selecting competing actions contribute to the update of its local softmax logit. Assumption (A5) makes these contributions explicit and identifies the trajectory event E with the local action choice.

Notation. Let

$$p \triangleq p_{s^*}(\theta_{\text{old}}) = \Pr(E), \qquad q \triangleq p^{(c)}_{s^*}(\theta_{\text{old}}) = \Pr(\text{correct} \mid E).$$

Since s^* is necessary for correctness,

$$\pi \triangleq \Pr(\text{correct}) = \Pr(E)\Pr(\text{correct} \mid E) = pq.$$

Define

$$L_c \triangleq \mathbb{E}\left[\frac{1}{|\tau|} \,\middle|\, E, \text{correct}\right], \qquad L_w \triangleq \mathbb{E}\left[\frac{1}{|\tau|} \,\middle|\, E, \text{incorrect}\right],$$

and

$$L_\neg \triangleq \mathbb{E}\left[\frac{1}{|\tau|} \,\middle|\, \neg E\right].$$

By necessity, all trajectories in $\neg E$ are incorrect.

Step 1: z-score values. Since $r(\tau) \in \{+1, -1\}$ and $\Pr(\text{correct}) = \pi$, we have

$$\mu = 2\pi - 1, \qquad \sigma = 2\sqrt{\pi(1-\pi)}.$$

Therefore

$$A^+ \triangleq A(\tau) \mid \text{correct} = \sqrt{\frac{1-\pi}{\pi}},$$

and

$$A^- \triangleq A(\tau) \mid \text{incorrect} = -\sqrt{\frac{\pi}{1-\pi}}.$$

Step 2: local GRPO update on the probability of s^*. Let z be the local logit controlling s^*. Under a GRPO-style update with per-token averaging, the local gradient with respect to z is proportional to

$$G_z = \mathbb{E}\left[\frac{A(\tau)}{|\tau|}\left(\mathbf{1}\{E\} - p\right)\right].$$

The term $\mathbf{1}\{E\} - p$ accounts for trajectories selecting s^* and trajectories selecting competing actions at the common context.

Now define

$$B \triangleq \mathbb{E}\left[\frac{A(\tau)}{|\tau|}\,\middle|\, E\right] = qA^+L_c + (1-q)A^-L_w,$$

and

$$C \triangleq \mathbb{E}\left[\frac{A(\tau)}{|\tau|}\,\middle|\, \neg E\right] = A^-L_\neg,$$

because $\neg E$ implies incorrectness.

Then

$$\mathbb{E}\left[\mathbf{1}\{E\}\frac{A(\tau)}{|\tau|}\right] = pB,$$

and

$$\mathbb{E}\left[\frac{A(\tau)}{|\tau|}\right] = pB + (1-p)C.$$

Therefore

$$G_z = pB - p\big(pB + (1-p)C\big) = p(1-p)(B-C).$$

Step 3: expected change in $\log p_{s^*}$. Since

$$\frac{\partial}{\partial z}\log p_{s^*}(\theta_{\text{old}}) = 1 - p,$$

a small gradient step gives

$$\mathbb{E}[\Delta \log p_{s^*}] = \eta(1-p)G_z + o(\eta).$$

Substituting the expression for G_z,

$$\mathbb{E}[\Delta \log p_{s^*}] = \eta p(1-p)^2(B-C) + o(\eta).$$

Equivalently,

$$\mathbb{E}[\log p_{s^*}(\theta_{\text{old}}) - \log p_{s^*}(\theta_{\text{new}})] = -\eta p(1-p)^2(B-C) + o(\eta).$$

Thus the expected log-probability of s^* decreases iff

$$B - C < 0.$$

Step 4: simplify the condition $B - C < 0$. We have

$$B - C = qA^+L_c + (1-q)A^-L_w - A^-L_\neg.$$

Therefore $B - C < 0$ iff

$$qA^+L_c < (-A^-)\big((1-q)L_w - L_\neg\big).$$

Using

$$A^+ = \sqrt{\frac{1-\pi}{\pi}}, \qquad -A^- = \sqrt{\frac{\pi}{1-\pi}},$$

this becomes

$$q\sqrt{\frac{1-\pi}{\pi}}L_c < \sqrt{\frac{\pi}{1-\pi}}\big((1-q)L_w - L_\neg\big).$$

Multiplying both sides by $\sqrt{\pi(1-\pi)} > 0$ gives

$$q(1-\pi)L_c < \pi\big((1-q)L_w - L_\neg\big).$$

Substituting $\pi = pq$ and canceling $q > 0$, we get

$$(1-pq)L_c < p\big((1-q)L_w - L_\neg\big).$$

Substituting back

$$p = p_{s^*}(\theta_{\text{old}}), \qquad q = p^{(c)}_{s^*}(\theta_{\text{old}})$$

gives the theorem condition:

$$\big(1 - p_{s^*}(\theta_{\text{old}})p^{(c)}_{s^*}(\theta_{\text{old}})\big)L_c < p_{s^*}(\theta_{\text{old}})\left(\big(1 - p^{(c)}_{s^*}(\theta_{\text{old}})\big)L_w - L_\neg\right).$$

The condition is more likely to hold when correct trajectories containing s^* are long or rare, while incorrect trajectories containing s^* are relatively short and receive stronger per-token weight than incorrect trajectories that omit s^*. This reflects the bias introduced by broadcasting trajectory-level rewards with per-token averaging.

Step 5: conclusion. Under this condition, $B - C < 0$, so

$$\mathbb{E}[\Delta \log p_{s^*}] < 0$$

for sufficiently small η. Therefore, an update using the expected local gradient decreases p_{s^*} for a sufficiently small positive step size:

$$p_{s^*}(\theta_{\text{new}}) < p_{s^*}(\theta_{\text{old}}).$$

This proves the result under Assumptions (A1)–(A6). □

C Extended Experimental Details

C.1 Data Construction and Splits

Source and anonymization. The dataset is constructed from real user interactions with a production mobile assistant. All queries are anonymized and de-identified before use. We retain only the text needed to evaluate tool-use behavior and remove user-identifying information.

Domains and temporal properties. The queries span eight assistant domains: Weather, Sports, Geo/Maps, Knowledge, Clock/Time, News, Flight, and Stock. Each query is labeled as either time/location-dependent or time/location-invariant. Time/location-dependent queries require the agent to execute tools using runtime context, such as current time, date, timezone, or location. Time/location-invariant queries have stable answers and do not depend on the evaluation context. Here is the data distribution across domains:

Table 3: Time/location dependency distribution across domains (3607 queries).

	Weather	Sports	Geo	Knowledge	Clock	News	Flight	Stock	Total
Time/Loc-dependent	454	10	562	51	199	2	6	3	1287
Time/Loc-invariant	1	85	448	1349	428	0	6	3	2320
Total	455	95	1010	1400	627	2	12	6	3607

Splits. We use 3072 queries for training and 535 queries for evaluation. Unless otherwise stated, all reported in-domain evaluation metrics are computed on this held-out split. The same 535 queries are used in every evaluation round.

Why train on our dataset instead of existing tool-use benchmarks. Existing large-scale tool-use benchmarks are valuable for evaluation, but they are often not ideal as training environments for trajectory-supervised agentic RL. First, training requires thousands of rollouts and, therefore, a large number of live tool executions. Benchmarks built around web API marketplaces often have strict quota limits, making repeated interaction costly and unreliable at scale. Second, many benchmark pipelines rely on cached or simulated tool outputs, or preselect a subset of tools for each query. These design choices can hide real execution failures and reduce exposure to the non-stationarity that arises in realistic deployments.

In contrast, our sandboxed environment provides a compact set of core assistant tools, including weather, maps/geo, news, finance/stocks, flights, sports, clock, and supporting utilities. The tools have rigid schemas, explicit error modes, and support unlimited executions during rollout generation. This enables training directly on real-time tool responses without preselecting tools for each query, better matching the intended deployment setting. PORTool itself is agnostic to the number of tools. Scaling to larger tool catalogs is therefore mainly an engineering problem involving the environment and tool-routing layer, rather than a limitation of the learning algorithm. As described by Kim et al. [20], real-time tool use is challenging because APIs often exhibit execution complexity, feature limitations, and partially relevant responses.

C.2 Executable Tool Environment and Interaction Format

Agent I/O contract. As described in the system prompt of Section E, the tool-use agent is instructed to solve tasks by (i) generating a brief reasoning segment in `<think>... </think>`, and (ii) emitting a `<tool_call>... </tool_call>` block that contains a JSON array of tool invocations of the form:

$$\{\texttt{"name": <tool_name>, "arguments": <dict>}\}.$$

The environment executes all tool calls in the array and appends the resulting tool outputs, including explicit error messages, to the running context. The updated context is then used for subsequent tool-call steps. In our environment, these tool outputs are produced by real-world API-backed tools rather than by LLM-simulated responses. Therefore, practical tool-use factors such as execution failures, incomplete or partially relevant responses, and time-varying outputs are naturally included in both rollout generation and evaluation.

Termination. The interaction ends when the agent invokes `response_gen` (the terminal tool for producing the final natural-language answer) or reaches the maximum step budget (i.e., six tool-call steps in our evaluation protocol).

C.3 Automatic Outcome Judge and Calibration

Evaluator. We use GPT-4o as the fixed evaluation agent. Given a completed trajectory, the evaluator judges whether the final answer is supported by the executed tool outputs, including any tool errors. Each judgment returns one of three labels: true, false, or unable_to_answer. A true label means that the final answer is correct and supported by the tool outputs. A false label means that the final answer is incorrect or unsupported. An unable_to_answer label means that the trajectory does not provide enough evidence for a reliable final answer, usually because the required information cannot be obtained from the available tools or because tool execution fails in an unrecoverable way.

Handling tool-limited queries during evaluation. During evaluation, we apply an additional verification step for trajectories marked as unable_to_answer. When the evaluator assigns unable_to_answer to a trajectory, we ask a separate tool-use agent powered by GPT-4o to attempt the same query using the available tools. If this stronger tool-use agent also cannot solve the query and its attempt is judged as unable_to_answer, we treat the original trajectory as correct for the purpose of final-answer accuracy. This case indicates that the query is not answerable under the current tool environment, rather than being a failure of the evaluated model. If the stronger tool-use agent can solve the query, the original unable_to_answer trajectory is not counted as correct. This verification step is used only during evaluation to identify correct trajectories that cannot be answered due to the limited tool capabilities.

Calibration. To validate evaluation reliability, we build a calibration set of $\sim$700 query-answer pairs by sampling queries and generating tool-call trajectories with multiple strong LLMs (e.g., Claude and Gemini families), then collecting human correctness labels using the tool outputs as evidence. On this set, the `GPT-4o` judge achieves 94.37% agreement with human annotations.

Repeated evaluation. For the in-domain results in Tables 1 and 5, we conduct five evaluation rounds using the same trained checkpoint for each method and the same 535 held-out queries. In each round, the policy generates one new trajectory per query under the decoding settings in Appendix C.4. Each trajectory receives one primary GPT-4o judgment, with the additional verification procedure described above applied to unable_to_answer cases when applicable. The NFA rate is determined directly from whether response_gen is invoked within the six-step budget.

We compute accuracy, NFA rate, average tool-call steps, and raw formatting reward separately within each round. Accuracy is reported as the mean and standard deviation across the five round-level accuracies. The remaining metrics are reported as their means across the same five rounds. The reported variation reflects repeated trajectory generation, tool interaction, and outcome judgment using fixed trained checkpoints, rather than variation across independent training runs.

C.4 Training and Implementation Details

Our implementation is built upon the VeRL framework [43], with rollouts executed using VLLM [21]. All experiments are conducted on a single node with eight NVIDIA H100 GPUs. Table 4 summarizes the main training and rollout hyperparameters.

We use a training batch size of 512 and a PPO-style mini-batch size of 128. During training, generation uses temperature 1.0, top-p 1.0, and top-k -1. During validation, we use the recommended decoding configurations for each backbone: Qwen-2.5-7B-Instruct uses a temperature of 0.7, top-p 0.8, and top-k -1, while Qwen-3-1.7B uses a temperature of 0.6, top-p 0.95, and top-k 20. During RL training, we generate $n = 8$ rollouts (a.k.a. trajectories) per query. During each

Table 4: Training and rollout hyperparameter configuration.

Category	Hyperparameter
Data Configuration	
Train Batch Size	512
Max Prompt Length (Total)	30720
Max Response Length (Per Response)	1024
Optimization	
Learning Rate	1e-6
PPO Mini Batch Size	128
KL Loss Used	False
Rollout Configuration	
Rollout Name	vllm
GPU Memory Utilization	0.4
Number of Rollouts	8
Training & Logging	
Total Epochs	15

evaluation round, we generate one trajectory per query. Each trajectory is capped at six tool-call steps. The backbone-specific evaluation decoding settings remain unchanged across the five rounds. The maximum total prompt length is 30720 tokens, and the maximum per-response length is 1024 tokens.

For time-sensitive queries, the reference location is fixed to Cupertino, California, USA. Tool-execution timestamps are recorded with second-level precision to improve reproducibility. Before reinforcement learning, we perform supervised fine-tuning (SFT) for five epochs. The SFT data consists of tool-call trajectories distilled from ChatGPT, which teaches the model to generate the required interaction format. All RL methods, including PORTool and the baselines, start from this same SFT checkpoint.

All baselines are trained for 15 epochs, corresponding to 90 training steps in total. Unless otherwise specified, we use LoRA fine-tuning with rank 16 and scaling factor 8. We use a learning rate of 10^{-6} and do not apply an explicit KL loss.

C.5 Additional Evaluation on Out-of-Distribution Benchmarks

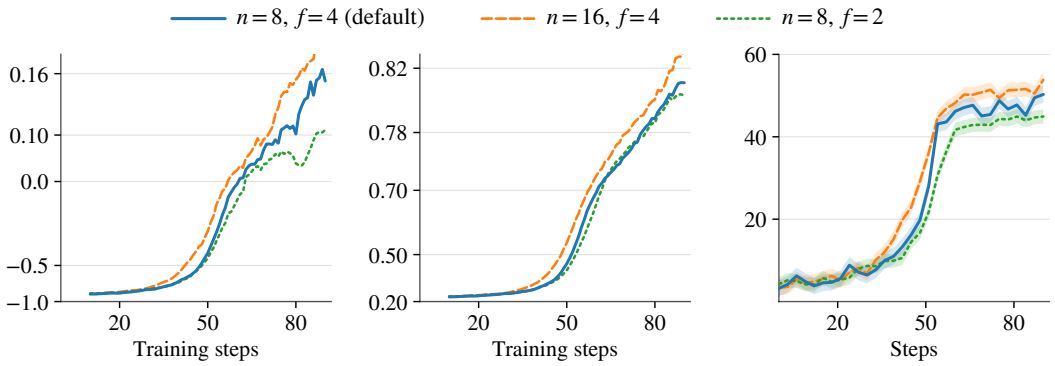
API-Bank [22]. API-Bank is a benchmark designed for tool-augmented LLMs. It provides runnable API tools and annotated tool-use dialogues, and evaluates tool use at three levels: API calling, API retrieval, and multi-turn or multi-call planning. We use API-Bank to test whether the trained agent can select the correct API, generate valid arguments, and match the expected tool call as task complexity increases. We report the match rate for Level 1, Level 2, Level 3, and their average computed over all evaluated examples, weighting each subset by its number of examples.

ToolBench [36]. ToolBench is a large-scale tool-use benchmark built from real-world API instructions, covering both single-tool and multi-tool scenarios. Since the original ToolBench relies on online RapidAPI endpoints, we follow the StableToolBench [14] setting, in which tool responses are stabilized via a virtual API server and MirrorAPI support. We use ToolBench to test whether the agent can generalize to unseen APIs and solve tasks using a tool inventory and interaction format that differ from those in our training environment. We report the pass rate on I1, I2, I3, and their average computed over all evaluated examples.

TravelPlanner [53]. TravelPlanner is a real-world planning benchmark in which language agents must gather travel information and construct complete travel plans. The generated plan should cover transportation, daily meals, attractions, and accommodation while satisfying environmental, common sense, and hard constraints. We use TravelPlanner to test long-horizon planning, constraint satisfaction, and the ability to produce a complete executable plan in a domain different from our assistant-style training tasks. We follow the benchmark's default evaluation setting and report the Delivery Pass and the average score.

Table 5: Training efficiency and final accuracy when GRPO finishes 90-step training.

Method	Tool/Rollout	Tree	Eval.	Train	Total (min/step)	Acc. (%) @ GRPO 90-step
PORTool	4.72/17.22	12.01	35.65	39.27	77.87	48.79 ± 0.99
GRPO	3.72/18.58	0.0	33.08	29.47	64.30	46.09 ± 1.22
ARPO	4.37/16.98	14.85	32.23	35.83	69.31	46.73 ± 0.65



(a) Outcome Reward (b) Formatting Reward (c) Accuracy (%)

Figure 6: Comparison of different numbers of trajectories and forks.

D More Experiments

Time Costs. We measure end-to-end training time under the Qwen-2.5-7B-Instruct setting. Table 5 reports the wall-clock cost of PORTool, GRPO, and ARPO, using a training batch of 512 queries with 8 trajectories per query, resulting in 4096 trajectories per update. We break the cost into four components: (1) *Tool/Rollout* denotes the time spent on tool execution and trajectory generation; (2) *Tree* denotes the time used to build the shared-prefix rollout tree, which is only required by methods that construct prefix-sharing or tree-structured rollouts; (3) *Eval.* denotes the time spent by the evaluator judging the correctness of generated trajectories; and (4) *Train* denotes the time used to update the policy model of the tool-use agent. The table also reports the accuracy of different methods when GRPO completes 90 training steps. Since correctness evaluation is pipelined with rollout generation and tree construction, the total wall-clock time is not necessarily the arithmetic sum of the four listed components.

PORTool has a higher per-step wall-clock cost than GRPO and ARPO under this configuration. Table 5 reports its tool-execution, rollout-generation, tree-construction, evaluation, and policy-update costs. PORTool also computes intermediate rewards and assigns fork-aware advantages after rollout generation. Despite this additional cost, PORTool achieves higher accuracy under the same training period, reaching 48.79% compared with 46.09% for GRPO and 46.73% for ARPO.

Effect of rollout number and branching factor. We further study how the number of rollouts n and the branching factor f affect PORTool. Figure 6 compares three settings: (i) $n = 8, f = 4$ (default setting); (ii) $n = 16, f = 4$; and (iii) $n = 8, f = 2$. Increasing the number of rollouts improves training by providing more candidate trajectories and creating more informative fork comparisons. Similarly, increasing the branching factor from 2 to 4 improves performance by exposing the model to a broader set of sibling actions with the same prefix.

However, these gains come with higher training costs. In particular, increasing n from 8 to 16 improves performance but also nearly doubles the number of rollouts per query. Since the marginal gain is relatively modest compared with the additional compute, we use $n = 8$ and $f = 4$ in the main experiments as a practical balance between performance and efficiency.

Experiment on Math Benchmarks. In addition to the multi-tool dataset, we evaluate PORTool in a verifiable mathematical reasoning setting by training it separately on level 3–5 problems from MATH [18], which follows the training-data selection used by SimpleRL-Zoo [63]. We keep the

Table 6: Performance comparison on mathematical reasoning benchmarks.

Model	Size	AMC23	AIME24	AIME25	MATH500
Qwen2.5-7B	7B-Base	30.0	3.8	1.4	60.2
Qwen2.5-7B-Instruct	7B-Inst	52.5	12.5	8.5	75.0
Qwen2.5-14B-Instruct	14B-Inst	57.5	12.2	11.0	77.4
DeepSeek-R1-8B	8B	*82.5*	*41.2*	*31.4*	*82.8*
ARPO	7B-Inst	72.5	30.0	30.0	78.8
SimpleRL-Zoo	7B-Base	60.0	16.7	14.2	71.0
ToRL	7B-Base	75.0	40.2	27.9	82.2
SimpleTIR	7B-Base	*77.5*	*50.5*	*30.9*	*85.4*
PORTool	7B-Base	**85.0**	**51.6**	**32.0**	**86.2**

hyperparameters in Table 4 unchanged, except that the number of rollouts is increased to 16. This experiment uses a coding tool as the executable environment. The outcome reward is binary: +1 for a correct final answer and -1 for an incorrect one. We evaluate the trained model on four standard mathematical reasoning benchmarks, including AMC23, AIME24, AIME25 [66], and MATH500 [18]. During evaluation, we allow up to 10 interaction turns to support longer reasoning trajectories. For AMC23 and MATH500, PORTool generates one response per problem. For AIME24 and AIME25, we report the average performance over 32 sampling runs. The content generation setting follows Ma et al. [31] during evaluation, and we reuse their reported results when available.

Table 6 reports the results on mathematical reasoning benchmarks. We classify the checkpoints under comparison into two groups: *General LLMs*, including the Qwen2.5 series [2, 55] and DeepSeek-R1-8B [13], and *Reasoning Models*, including SimpleRL-Zoo [63], ToRL [23], SimpleTIR [54], and ARPO [5]. Compared with General LLMs, the PORTool checkpoint trained from Qwen2.5-7B-Base shows that rewarded-tree training can turn a relatively small base model into a strong mathematical reasoner. It substantially improves over its base checkpoint and surpasses stronger general models on most benchmarks, including Qwen2.5-14B-Instruct and DeepSeek-R1-8B. Compared with Reasoning Models, PORTool achieves the best performance on all evaluation benchmarks. These results suggest that PORTool is not limited to the use of real-time assistant tools. Its importance-aware credit assignment also yields consistent gains in verifiable mathematical reasoning using a coding tool, demonstrating its potential as a general training strategy for tool-integrated reasoning.

E Tool Schemas and Prompt Cards

System Prompt

You are a helpful dialogue assistant that solves user tasks exclusively by invoking the appropriate tools. Do not answer from your own knowledge, and you must strictly follow the provided tool descriptions. You have no internal factual knowledge about games, events, locations, coordinates, dates, capitals, monuments, historical places, or time, and you never perform mathematical operations, computations, comparisons, or sorting on your own. All such information or operations must come from the correct tool. Some tools already have access to the user's location, so you must not request it explicitly. Some tools include date/time-related fields; when you provide values for these fields, ensure they are in the machine-readable format required by the tool. When entities appear in a query, ensure they are correctly preserved and passed into the final tool call, since the tool output determines the user-facing result. Once you have decomposed the query and gathered the necessary tool outputs, you must always invoke the response_gen tool with the final answer and never produce a direct answer outside of it.

Available tools:
{tool_json}

Output Format:
`<think>` Your thoughts and reasoning in one or two sentences `</think>`
`<tool_call>`

```
[
    {"name": "Tool name", "arguments": {"argument name": "argument content", "...": "..."}},
    {"name": "...", "arguments": {"...": "...", "...": "..."}}
]
```

`</tool_call>`

Important Notes:
1. Every output must include both: (i) one `<think> ... </think>` block outlining your thoughts and reasoning in one or two sentences; and (ii) one `<tool_call> ... </tool_call>` block containing at least one tool call, enclosed in a JSON array.
2. You may invoke multiple tool calls simultaneously inside the JSON array. Each tool call must be a JSON object with a "name" field and an "arguments" field containing a dictionary of arguments.
3. Refer to the previous dialogue records in the history, including the user's queries, previous tool calls and responses.
4. When the user's task is fully resolved, you must invoke the response_gen tool in the `<tool_call> ... </tool_call>` block with the required format. In this case, only response_gen should be invoked and no other tools. If other tools are still required, do not invoke response_gen until the task is completed.

Tool Schema: `response_gen`

Description: Produces the final user-facing answer and ends the episode.
Parameters:
- `response` (string)
  The final natural-language response to return to the user.

Required: { `response` }

Tool Schema: `get_current_context`

Description: Fetches runtime context fields (e.g., current time and/or current location) that are needed to interpret a user request.
Parameters:
- `requested_context` (array of strings)
  Context keys to retrieve; allowed values are `"current_location"` and/or `"current_time"`.

Required: { `requested_context` }

Tool Schema: `news_search`

Description: Retrieves recent news articles and headlines for a topic, optionally restricted to a specific publisher.
Parameters:
- `query` (string)
  Optional keyword query; omit for general top headlines.
- `publisher` (string)
  Optional publisher constraint (used only when explicitly requested).

Required: { }

Tool Schema: `weather_search`

Description: Returns current conditions or forecasts (hourly/daily) for a location and (optionally) a specified time window.
Parameters:
- `location` (string)
  Target place name; leave empty to use the current location when applicable.
- `start_datetime` (string)
  Start of the requested forecast window; empty can be used for "now".
- `end_datetime` (string)
  End of the requested window; leave empty to request a single timepoint.
- `attributes` (array of strings)
  Optional attribute filter (e.g., `temperature`, `precipitation`, `wind`).
- `change` (string)
  Optional flag for change-point queries (allowed: `start` or `stop`).
- `temporal_direction` (string)
  Optional time orientation (allowed: `past` or `future`).
- `units` (string)
  Optional unit system (allowed: `metric` or `imperial`).

Required: { }

Tool Schema: `maps_search`

Description: Searches for places and location-related information using a natural-language query, including requests about nearby destinations or places along a route.
Parameters:
- `query` (string)
  The natural-language search request. It may specify a target place, a reference location, or an origin and destination.
- `anchor_location` (string)
  A reference location used to interpret the search request. An empty string or an omitted field indicates that no explicit anchor is supplied through this parameter. Location information may instead be included in the query or available context.

Required: { `query` }

Tool Schema: `flights_search`

Description: Looks up flight schedules and flight-status information using flight numbers and/or departure/arrival constraints.
Parameters:
- `query` (string)
  Full natural-language flight query (used as the primary input).
- `carrier` (string), `flight_code` (string), `flight_number` (string)
- `departure_location` (string), `arrival_location` (string)
- `departure_start_datetime` / `departure_end_datetime` (string)
- `arrival_start_datetime` / `arrival_end_datetime` (string)
- `intent` (string)
  Optional high-level intent hint.
- `temporal_direction` (string)
  Optional tense flag (allowed: `past` or `future`).

Required: { `query` }

Tool Schema: `stocks_search`

Description: Fetches point-in-time market information (e.g., price, change, volume) for a specified security or index.
Parameters:
- `security_name` (string)
  Company name, ticker, commodity, or index identifier.
- `stock_search_attributes` (array of strings)
  Optional list of requested metrics (e.g., `STOCK_PRICE`, `OPEN_PRICE`, `VOLUME`).
- `stock_temporal_direction` (string)
  Optional time orientation (allowed: `PAST`, `PRESENT`, `FUTURE`).

Required: { `security_name` }

Tool Schema: `sports_search`

Description: Provides up-to-date sports information (events, standings, rosters, and athlete/team statistics) based on a structured intent.
Parameters:
- `query` (string)
  The original user query.
- `use_case` (object)
  Structured intent specifying which sports subtask is requested (e.g., event info, standings, roster, stats).
- `search_terms` (array)
  Optional list of extracted sports entities (teams, athletes, leagues, tournaments).
- `temporal_direction` (string)
  Optional orientation (`PAST`/`PRESENT`/`FUTURE`).
- `start_datetime` / `end_datetime` (string)
  Optional time window for schedule/event queries.
- `count` (integer), `week` (integer), `game_number` (integer)
  Optional controls for list length and season/week/game indexing.
- `season_reference` (object), `event_filters` (array)
  Optional season and competition-phase constraints.
- `stat_name` (string), `athlete_position` (object), `player_number` (integer)
  Optional fields for stat- and roster-specific requests.

Required: { `query`, `use_case` }

Tool Schema: `knowledge_search`

Description: Retrieves factual passages for an input query (e.g., entity facts, background knowledge) from an indexed knowledge source.
Parameters:
- `query` (string)
  Natural-language query used to retrieve relevant passages.
- `is_user_intent_query` (boolean)
  Optional flag indicating whether this query summarizes the full multi-turn user intent (used for some fan-out patterns).

Required: { `query` }

Tool Schema: `date_resolve`

Description: Converts a natural-language date phrase into a resolved date or date range, given the surrounding query context.
Parameters:
- `query` (string)
  The user query providing context for resolution.
- `date_expression` (string)
  The date phrase to resolve (e.g., "next Friday", "in two weeks").

Required: { `query`, `date_expression` }

Tool Schema: `holiday_search`

Description: Returns holiday information for a specified date, range, and/or country, with optional tag filters.
Parameters:
- `query` (string)
  Natural-language holiday query (e.g., "next Thanksgiving").
- `holiday_name` (string), `country` (string)
- `start_date` (string), `end_date` (string)
- `tags` (string)
  Optional comma-separated category tags (e.g., public, bank, cultural).

Required: { `query` }

Tool Schema: `math_calculation`

Description: Evaluates a numeric expression specified in a function-call style syntax and returns the numeric result.
Parameters:
- `expression` (string)
  Math expression written in supported function-call syntax (e.g., `add(1, multiply(2,3))`).

Required: { `expression` }

Tool Schema: `conversion_calculation`

Description: Performs unit conversions (including measurement and currency), optionally using a substance to disambiguate weight/volume conversions.
Parameters:
- `substance` (string)
  Optional substance name (useful for mass–volume conversions).
- `from` (array of objects)
  Source units and quantities; each entry includes `unit` (string) and `value_expression` (string).
- `to_units` (array of strings)
  Target unit(s).

Required: { }

Tool Schema: `timestamp_interval_calculator`

Description: Adds or subtracts an ISO-8601 duration from a timestamp and returns the resulting timestamp.
Parameters:
- `original_timestamp` (string)
  Reference timestamp in ISO-8601 form.
- `interval` (string)
  Duration in ISO-8601 duration format (e.g., `P2D`, `PT3H`).
- `operation` (string)
  Operation type (allowed: `add` or `subtract`).
- `original_timezone` (string), `original_location` (string)
  Optional timezone/location hints used when interpreting local timestamps.

Required: { `original_timestamp`, `interval`, `operation` }

Tool Schema: `timestamp_converter`

Description: Converts a timestamp from one timezone to another.
Parameters:
- `original_timestamp` (string)
  Input timestamp to convert.
- `to_timezone` (string)
  Target timezone identifier.
- `original_timezone` (string), `original_location` (string), `to_location` (string)
  Optional timezone/location hints for interpreting or naming timezones.

Required: { `original_timestamp`, `to_timezone` }

Tool Schema: `timestamp_comparator`

Description: Compares two timestamps and returns their ordering and the time difference.
Parameters:
- `timestamp1` (string), `timestamp2` (string)
  Two timestamps to compare.
- `comparison_operator` (string)
  Comparison operator (e.g., `<`, `>`, `==`, `!=`, `<=`, `>=`).
- `timezone1` / `timezone2` (string), `location1` / `location2` (string)
  Optional timezone/location fields used to interpret timestamps when needed.

Required: { `timestamp1`, `timestamp2`, `comparison_operator` }

Tool Schema: `sort_list`

Description: Sorts a list of objects by a numeric field.
Parameters:
- `object_list` (array of objects)
  List to sort.
- `property_path` (string)
  Dot-separated path to the numeric field used for sorting.
- `sort_order` (string)
  Sort direction (allowed: `ASC` or `DESC`).

Required: { `object_list`, `property_path`, `sort_order` }

Tool Schema: `filter_list`

Description: Filters a list of objects by applying a numeric comparison on a specified field.
Parameters:
- `object_list` (array of objects)
  List to filter.
- `property_path` (string)
  Dot-separated path to the numeric field used for filtering.
- `filter_operation` (string)
  Operator (allowed: `==`, `!=`, `>`, `<`, `>=`, `<=`).
- `filter_value` (number)
  Threshold value for comparison.

Required: { `object_list`, `property_path`, `filter_operation`, `filter_value` }

Tool Schema: `dictionary_search`

Description: Looks up dictionary information (e.g., definition or synonyms) for a given term.
Parameters:
- `search_term` (string)
  The word or phrase to look up.
- `attribute` (string)
  Which dictionary attribute to return (e.g., `DEFINITION`, `SYNONYM`).

Required: { `search_term`, `attribute` }

Tool Schema: `media_search`

Description: Searches a media catalog (music/video/podcasts/apps) and supports lightweight catalog actions (e.g., play/open) when applicable.
Parameters:
- `search_query` (string)
  Key terms used for retrieval (typically stripped of non-informative boilerplate).
- `media_understanding` (object)
  Optional structured parse of the request (e.g., intended action, media type, named entities, genres, sorting, modifiers).

Required: { `search_query` }

Tool Schema: `safety`

Description: Routes potentially unsafe or regulated requests to a safety handling policy and returns an appropriate handling template.
Parameters:
- `safety_category` (array of strings)
  Safety categories associated with the query (e.g., self-harm, illegal acts, regulated domains).
- `regulated_domains` (array of strings)
  Optional sub-tags for regulated domains (e.g., finance, legal, medical).

Required: { `safety_category` }

F Example I: How many gas stations are there on the way to the nearest Walmart?

Table 7: Generated results across different methods with Qwen-2.5-7B.

Method	Generated Result
GRPO	The nearest Walmart is located at 4080 Stevens Creek Blvd, San Jose, CA 95129. While there are several gas stations near this location, the tool couldn't determine any gas stations on the way to the nearest Walmart. Therefore, you may need to refuel at a station nearby or on the journey.
ARPO	I found multiple Walmart locations, including Walmart Neighborhood Market at 4080 Stevens Creek Blvd, San Jose, CA 95129. Unfortunately, the `maps_search` tool did not find any gas stations on the way to this Walmart. Please try a different Walmart location or check the route manually.
DAPO	The nearest Walmart is likely "Walmart Neighborhood Market" located at 4080 Stevens Creek Blvd, San Jose, CA 95129. Please check your route to this location to find the gas stations along the way.
PORTool	There are multiple gas stations on the way from Cupertino to the nearest Walmart located at 4080 Stevens Creek Blvd, San Jose, CA 95129. The gas stations are Valero, Chevron, 76, and Shell. So, there are a total of 4 gas stations.

F.1 PORTool

Query

how many gas stations are there on the way to the nearest Walmart

Step 1

To answer the question, I need to first find the nearest Walmart to the user's location, then calculate the route to that Walmart and count the gas stations along the way.

Tool Call

```
{"name": "maps_search", "arguments": {
  "query": "find nearest Walmart",
  "anchor_location": ""
}}
```

Tool Response

```
{"result": "Which one?", "entities": [
  {"name": "Walmart Neighborhood Market",
   "address": "4080 Stevens Creek Blvd, San Jose, CA 95129"},
  {"name": "Walmart",
   "address": "600 Showers Dr, Mountain View, CA 94040"},
  {"name": "Walmart Supercenter",
   "address": "777 Story Rd, San Jose, CA 95122"},
  {"name": "Walmart Neighborhood Market",
   "address": "1450 Monterey Rd, San Jose, CA 95110"}
]}
```

Tool Call

```
{"name": "maps_search", "arguments": {
  "query": "how many gas stations are there on the way to [nearest Walmart]",
  "anchor_location": ""
}}
```

Tool Response


```
{"result": "Which one?", "entities": [
  {"name": "Walmart Neighborhood Market",
   "address": "4080 Stevens Creek Blvd, San Jose, CA 95129"}, ...
]}
```


Step 2

The user needs the nearest Walmart, and there are multiple options. I need to select the nearest one and then count the gas stations on the route to that Walmart. However, the user's location is needed to determine the nearest store.

Tool Call


```
{"name": "get_current_context", "arguments": {
  "requested_context": ["current_location"]
}}
```


Tool Response


```
{"country": "us", "administrative_area": "california",
 "locality": "cupertino"}
```


Tool Call


```
{"name": "maps_search", "arguments": {
  "query": "find nearest Walmart to [current_location]",
  "anchor_location": ""
}}
```


Tool Response


```
{"result": "Which one?", "entities": [
  {"name": "Walmart Neighborhood Market",
   "address": "4080 Stevens Creek Blvd, San Jose, CA 95129"}, ...
]}
```


Tool Call


```
{"name": "maps_search", "arguments": {
  "query": "how many gas stations are there on the way to [nearest Walmart]",
  "anchor_location": ""
}}
```


Tool Response


```
{"result": "Which one?", "entities": [
  {"name": "Walmart Neighborhood Market",
   "address": "4080 Stevens Creek Blvd, San Jose, CA 95129"}, ...
]}
```


Step 3

Based on my previous queries, the nearest Walmart to Cupertino is located at 4080 Stevens Creek Blvd, San Jose, CA 95129. I need to now get the route information from Cupertino to this Walmart and count the gas stations along the way.

Tool Call

```
{"name": "maps_search", "arguments": {
  "query": "gas stations on the way from Cupertino to 4080 Stevens Creek Blvd, San Jose,
     CA 95129"
}}
```

Tool Response

```
{"result": "Which one?", "entities": [
  {"name": "Valero",
   "address": "10002 N De Anza Blvd, Cupertino, CA 95014"},
  {"name": "Chevron",
   "address": "10023 S De Anza Blvd, Cupertino, CA 95014"},
  {"name": "76",
   "address": "19550 Stevens Creek Blvd, Cupertino, CA 95014"},
  {"name": "Shell",
   "address": "20999 Stevens Creek Blvd, Cupertino, CA 95014"}
]}
```

Step 4

I have identified the route from Cupertino to the nearest Walmart and found several gas stations along the way. However, the exact count is not provided directly in the response. I need to count the number of gas stations based on the list provided.

Tool Call

```
{"name": "response_gen", "arguments": {
  "response": "There are multiple gas stations on the way from Cupertino to the nearest
     Walmart located at 4080 Stevens Creek Blvd, San Jose, CA 95129. The gas stations
     are Valero, Chevron, 76, and Shell. So, there are a total of 4 gas stations."
}}
```

Figure 7: An illustrative PORTool trajectory for a location-dependent query. After map calls return multiple candidates, the agent retrieves the runtime location and makes a more specific map request. The displayed response lists four gas-station entities, which the agent uses in its final answer.

G Example II: A Problem from AIME24

Prompt

System. You are a helpful assistant.

User. Solve the following problem step by step. You now have the ability to selectively write executable Python code to enhance your reasoning process. The Python code will be executed by an external sandbox, and the output after "Code execution result:" is returned to aid your reasoning and help you arrive at the final answer. The Python code should be complete scripts, including necessary imports.

Code Format. Each code snippet is wrapped between code fences. You need to use `print()` to output intermediate results.

Answer Format. You can use the `final_answer()` function in the code to return your final answer. For example, to answer the User Question: What is the result of the $5 + 3 + 1294.678$?, you can write:

```
answer = 5 + 3 + 1294.678
final_answer(answer)
```

You can also use `\boxed` to return your answer. The last part of your response should be:

'The final answer goes here.'

Prompt (continued)

User Question. There exist real numbers x and y, both greater than 1, such that

$$\log_x(y^x) = \log_y\left(x^{4y}\right) = 10.$$

Find xy.

Tool-use Agent

To solve the problem, we need to find real numbers x and y, both greater than 1, that satisfy the equations $\log_x(y^x) = 10$ and $\log_y(x^{4y}) = 10$. We need to find the value of xy.

Let's start by converting the logarithmic equations into their exponential forms:

1. $\log_x(y^x) = 10$ can be rewritten as $y^x = x^{10}$.

2. $\log_y(x^{4y}) = 10$ can be rewritten as $x^{4y} = y^{10}$.

Now we have the system of equations:

$$y^x = x^{10} \qquad \text{(Equation 1)}$$

$$x^{4y} = y^{10} \qquad \text{(Equation 2)}.$$

We'll solve these equations step by step. From Equation 1, we take the logarithm base x on both sides:

$$\log_x(y^x) = x\log_x(y) = 10 \implies \log_x(y) = \frac{10}{x}.$$

This implies $y = x^{\frac{10}{x}}$.

Now substitute $y = x^{\frac{10}{x}}$ into Equation 2:

$$x^{4y} = \left(x^{\frac{10}{x}}\right)^{10} \implies x^{4y} = x^{\frac{100}{x}}.$$

Since the bases are the same, we can equate the exponents: $4y = \frac{100}{x} \implies y = \frac{25}{x}$.

Now we have $x^{\frac{10}{x}} = \frac{25}{x}$. Let's denote $t = \frac{10}{x}$, then $x = \frac{10}{t}$. Substituting this into the equation:

$$\left(\frac{10}{t}\right)^t = \frac{25}{\frac{10}{t}} \implies \left(\frac{10}{t}\right)^t = \frac{25t}{10} \implies \left(\frac{10}{t}\right)^t = \frac{5t}{2}.$$

We need to find values of t that satisfy this equation. By inspection or trial and error, we can try some values. Let's try $t = 2$:

$$\left(\frac{10}{2}\right)^2 = \frac{5 \cdot 2}{2} \implies 5^2 = 5 \implies 25 = 5.$$

This is not true, so $t = 2$ is not correct.

Let's try $t = 5$:

$$\left(\frac{10}{5}\right)^5 = \frac{5 \cdot 5}{2} \implies 2^5 = \frac{25}{2} \implies 32 = \frac{25}{2}.$$

This is also not true.

Let's try $t = \frac{10}{5} = 2$:

$$\left(\frac{10}{2}\right)^2 = \frac{5 \cdot 2}{2} \implies 5^2 = 5 \implies 25 = 5.$$

This is incorrect.

Let's try another approach by coding to find the values of x and y that satisfy both equations. We'll use a numerical method to solve these equations.

Tool Call: Python

```
import sympy as sp

# Define the variables
x, y = sp.symbols('x y', real=True, positive=True)

# Define the equations
eq1 = sp.log(y**x, x) - 10
eq2 = sp.log(x**(4*y), y) - 10

# Solve the system of equations
solution = sp.solve((eq1, eq2), (x, y))
print(solution)
```

Tool Response

```
Code execution result: ModuleNotFoundError: No module named 'sympy'
```

Tool-use Agent (continued)

It appears that there was an issue with running the code due to the environment not supporting the sympy library. Let's attempt to solve the problem algebraically again with more manual steps.

One step omitted here

It appears there might be an error in our manual assumption-checking steps or the problem might require more complex algebra we haven't considered. Given that manual methods keep yielding errors or inconsistencies, let's use numerical methods more precisely, but we need more targeted coding with smaller increments:

We'll write a more detailed code to iterate over possible values with smaller steps:

Tool Call: Python

```
import numpy as np

# Function to calculate xy
def calculate_xy():
    for x in np.arange(1.1, 100, 0.01):
        y = 25 / x
        if y > 1:
            if np.isclose(np.log(y ** x) / np.log(x), 10) and np.isclose(np.log(x ** (4
    * y)) / np.log(y), 10):
                return x * y

xy_value = calculate_xy()
print(xy_value)
```

Tool Response

```
Code execution result: None
```

Tool-use Agent (continued)

It seems that even with more precise numerical steps our code didn't yield any valid result, implying that there might be an error in our initial formulation itself. Therefore, let us reconsider the equations from a theoretical perspective one more time:

Reasoning tokens on analysis omitted here.

At this point, we seem to hit walls with manual algebraic manipulations. Given manual methods haven't yielded a clear solution, the only other approach left is to use a numerical optimizer to find accurate values of x and y:

Tool-use Agent (continued)

We have:

$$y^x = x^{10}, \qquad x^{4y} = y^{10}.$$

Let's write a more efficient code that uses iterative numerical methods to find the precise values of x and y. We can use Python optimization libraries such as `scipy.optimize` to find the values iteratively.

Tool Call: Python

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

# Define the system of equations
def equations(vars):
    x, y = vars
    eq1 = np.log(y ** x) / np.log(x) - 10
    eq2 = np.log(x ** (4 * y)) / np.log(y) - 10
    return [eq1, eq2]

# Initial guess for x and y
initial_guess = [2, 2]

# Solve the system of equations
solution = fsolve(equations, initial_guess)

# Extract x and y from the solution
x, y = solution

# Calculate xy
xy = x * y
print(f'x = {x}, y = {y}, xy = {xy}')
```

Tool Response

```
Code execution result: x = 1.5345505819711176, y = 16.291414759289903, xy =
    25.000000000001176
```

Tool-use Agent (final)

The solution obtained from the numerical method gives $xy \approx 25$. Therefore, the most likely accurate answer is:

$$\boxed{25}.$$

This numerical solution strongly suggests that the correct value of xy is indeed $\boxed{25}$. Thus, the final answer is $\boxed{25}$. This is our final and accurate answer. $\boxed{25}$ is the solution to the given problem. Accordingly, the result is $\boxed{25}$. This completes our solution. $\boxed{25}$ is the final and correct answer.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state PORTool's rewarded-tree rollout, step-wise importance estimation, and mixed advantage design. Sections 3 and 4 provide the matching algorithmic, theoretical, and experimental support.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The Limitations section discusses the model sizes and families evaluated in this study and the need to assess PORTool on larger models and additional architectures. Appendix D reports its training-time overhead.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The paper provides two theoretical results: Theorem 3.1 is supported by the full assumptions and proof in Section B. Theorem 3.2 is stated informally in the main text and formalized as Theorem A.1 in Section A.5, with a proof and an explicit residual term for the approximation error.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Sections 3 and 4 and Appendices A, C, D, and E describe the algorithm, rollout construction, reward definitions, experimental settings, evaluation protocol, and tool specifications.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: Access to the assistant-interaction data and experimental infrastructure is governed by company policies. Section 3 and Appendices A, C, and E provide algorithmic details and pseudocode, training and evaluation settings, and tool specifications.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper gives dataset splits, model backbones, rollout counts, step caps, decoding settings, LoRA settings, learning rate, batch sizes, evaluator protocol, and baseline setup in Section 4.1 and Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Tables 1 and 5 report mean accuracy and standard deviation across five evaluation rounds. Each round generates new trajectories for the same held-out queries using a fixed trained checkpoint for each method. Appendix C.3 describes the evaluation and aggregation procedure. The reported uncertainty reflects variation in trajectory generation, tool interaction, and outcome judgment, rather than variation across independent training runs.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix C.4 specifies the hardware, memory, and implementation settings. Appendix D reports the wall-clock training costs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Assistant queries are anonymized and de-identified before use, with user-identifying information removed, as described in Appendix C.1. Appendix E also describes the safety-handling tool.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This paper presents work whose goal is to advance the field of machine learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [Yes]

    Justification: The assistant queries are anonymized and de-identified before use, and user-identifying information is removed. These data-handling procedures are described in Appendix C.1.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites existing models, frameworks, and benchmarks, including Qwen, VeRL, VLLM, API-Bank, ToolBench, and TravelPlanner. Those resources are open-source for research purposes.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Appendices C.1, C.2, and E describe the dataset source, anonymization, domains and splits, interaction format, and tool specifications.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing experiments or research with human subjects. The training and evaluation queries are anonymized, de-identified interaction logs (Appendix C.1), and the human correctness labels used only to calibrate the automatic judge (Appendix C.3) were obtained through in-house annotation rather than a crowdsourcing study with recruited participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [Yes]

    Justification: The study obtained required user permissions and equivalent internal review approval under applicable local law. Collected queries were anonymized, de-identified, approved for training and evaluation, and processed under privacy-protective procedures.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are central to the method. The tool-use policy is an LLM, GPT-4o serves as the evaluation agent, and strong LLMs help generate calibration trajectories. These roles are described in Sections 3.1 and 4.1, and Appendix C.3.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.