PORTool: Tool-Use LLM Training with Rewarded Tree

Anonymous ACL submission

Abstract

Current tool-use large language models (LLMs) are trained on static datasets, enabling them to interact with external tools and perform multi-step, tool-integrated reasoning, which produces tool-call trajectories. However, these models imitate how a query is resolved in a generic tool-call routine, thereby failing to explore possible solutions and demonstrating limited performance when exposed to an evolved, dynamic tool-call environment. In this work, we propose PORTool, a reinforcement learning (RL) method that encourages a tool-use LLM to explore various trajectories yielding the correct answer. Specifically, this method starts with generating multiple rollouts for a given query, and some of them share common tool calls in the first few steps, thereby forming a tree-like structure. Next, based on these rollouts, we assign rewards to each step within a trajectory, prioritizing the correctness of the final answer while also considering the success of tool calls. For a step shared by multiple trajectories, we apply an adaptive strategy that reflects the likelihood that the step leads to a correct answer. Finally, these step rewards are used to calculate fork-relative advantages, blended with trajectory-relative advantages, to optimize the tool-use LLM. The experiments utilize 17 tools to address user queries, encompassing topics such as weather and navigation that require real-time information. We conduct ablation studies to systematically justify the necessity and the design robustness of step rewards. Furthermore, we compare the proposed PORTool with other training approaches, and it demonstrates a significant improvement in terms of final accuracy and the number of tool-call steps.

1 Introduction

As a representative paradigm for synergizing reasoning and action, the ReAct framework (Yao et al., 2023b) activates large language models (LLMs) to combine reasoning (step-by-step analysis) and acting (tool calls), achieving strong problem-solving performance. Early works (Yao et al., 2023a,b; Wei et al., 2022; Schick et al., 2023) enable this capability primarily through prompt engineering, in which carefully 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 interactive tool use*.

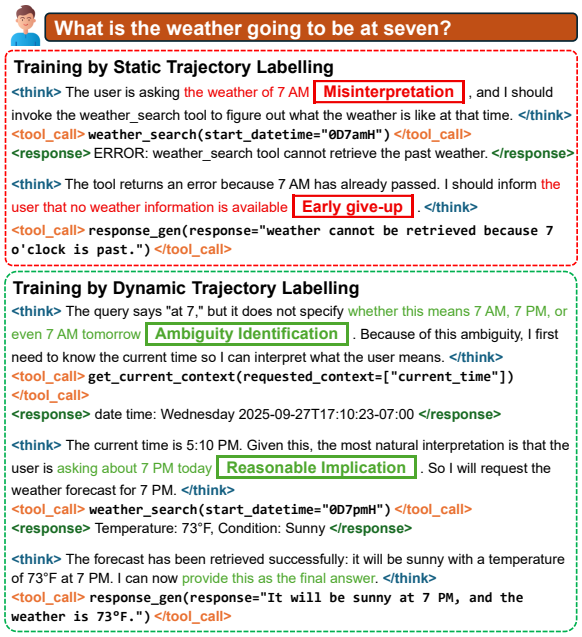


Figure 1: Training by labeled static tool-call trajectories cannot handle a real-time query.

Reinforcement learning (RL) has emerged as a promising direction to address this gap, leading to a growing number of research works (Qian et al., 2025; Zhang et al., 2025; Dong et al., 2025; Zeng et al., 2025; Jin et al., 2025; Li et al., 2025; Singh et al., 2025; Xue et al., 2025; Feng et al., 2025a; Wei et al., 2025; Lin and Xu, 2025). Most exist-

ing works concentrate on a narrow set of tools, such as web search or program compilers, which restricts the trained model's applicability to a wide range of function-like tools with rigid formatting requirements and specialized functionalities. ToolRL (Qian et al., 2025) and Tool-N1 (Zhang et al., 2025) broaden the scope by training LLMs to interact with diverse tool schemas. Their datasets consist of structured inputs, including queries, tool descriptions, and, if available, historical tool-call trajectories, paired with the expected tool calls for subsequent steps. Both methods adopt group-relative policy optimization (GRPO) (Shao et al., 2024) for fine-tuning, where the model generates various tool-call steps for each input and receives rewards based on consistency with the expected tool calls, thereby reinforcing tool-use capability.

These two approaches, however, face several limitations because they assume the existence of ground-truth tool calls at every intermediate step. First, constructing such datasets is economically impractical in real-world settings, as it requires substantial human effort to understand tool execution mechanisms and annotate each step. Second, this design encourages the model to imitate predefined tool calls. While effective for producing generic routines that solve common queries, it diminishes the model's ability to explore alternative trajectories. As a result, potentially valid solutions are either ignored or penalized for diverging from the expected calls. This limits the model's ability to handle real-time queries, as illustrated in Figure 1.

In this work, we begin by collecting a set of real user queries that require both factual and real-time information, along with a suite of executable tools. Instead of providing a generic tool-call routine, we build an agentic system that consists of a tool-use agent and an evaluation agent: The tool-use agent enables the LLM to interact with external tools to generate complete tool-call trajectories, while the evaluation agent assesses the correctness of each trajectory based on its final answer to the input query. With this system, we follow the existing tool-use training works (Jin et al., 2025; Qu et al., 2025; Dong et al., 2025) to reinforce the LLM on the tool-use agent through GRPO. Specifically, the model generates multiple tool-call trajectories independently, and GRPO computes the advantages across the trajectory level, assigning identical advantages to all tokens within each trajectory.

Apparently, GRPO neglects the varying contributions of each step to the final result. In particular, even an incorrect trajectory may contain informative intermediate steps that, if recombined differently, could bring a correct answer. However, such potential is lost when all steps are assigned with uniform rewards (Zeng et al., 2025). Conversely, certain steps within a correct trajectory may have no influence on the final result. These observations emphasize the need to *quantify each step's importance with the help of the evaluation agent*.

Toward the goal, we propose PORTool, a **P**olicy **O**ptimization algorithm with **R**ewarded Tree for **tool**-use LLMs. PORTool introduces *tree rollout*, in which certain tool-call steps branch into multiple trajectories and eventually lead to various subsequent tool calls. After evaluating each trajectory, the evaluation agent assigns step rewards that consider both the correctness of the final answer and the formatting issues of each step. Correctness is weighted more strongly in the reward signals, encouraging the model to favor steps that yield the correct answer even when the output format is erroneous. For shared steps across trajectories, we adopt an adaptive strategy to determine their rewards. Finally, with the step-wise reward, we optimize the LLM for generating a preferred tool-call step, blended with a trajectory-based reward that captures the holistic effectiveness of each trajectory.

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

- To the best of our knowledge, this is the first work to reinforce the LLM for tool-use capability by interacting with numerous executable tools, demonstrating evolved and dynamic outputs based on real-time information.
- We propose an algorithm named PORTool, which integrates an innovative tree rollout strategy and assigns rewards to any intermediate steps. thereby optimizing the LLM by capturing the effectiveness of the holistic trajectory and the preferred steps.
- We conduct ablation studies to justify the effectiveness of the proposed design of the step reward. In addition, we compare the proposed PORTool with other policy optimization algorithms and demonstrate superiority in terms of final accuracy and the number of tool-call steps.

2 Related Works

RL algorithms for enabling LLMs with reasoning. The rapid development of LLMs highlights the pivotal role of RL algorithms in training (Schulman et al., 2017; Rafailov et al., 2023; Ahmadian et al., 2024; Yu et al., 2025). Recent breakthroughs such as DeepSeek-R1 (Guo et al., 2025) disclose that RL algorithms can endow LLMs with chain-of-thought reasoning abilities (Wei et al., 2022) by assigning verifiable or rule-based rewards (Zhu et al., 2025; Yeo et al., 2025; Havrilla et al., 2024). This insight has spurred a wave of new policy optimization approaches (Yu et al., 2025; Zheng et al., 2025; Ahmadian et al., 2024; Liu et al., 2025; Chu et al., 2025b). In multi-turn tool-use scenarios, these approaches assign a single reward to the entire trajectory, which obscures the varying significance of individual steps and fails to recognize how intermediate tool calls shape the final outcome. To tackle the limitation, GiGPO (Feng et al., 2025b) assigns rewards at the step level and normalizes them across actions within the same state when training the policy model. However, this framework does not extend to our scenario, as the state information is not explicitly defined.

Among those RL works, the most related concurrent work is ARPO (Dong et al., 2025). ARPO proposes an entropy-based adaptive rollout mechanism that generates multiple rollouts forming a tree structure. Specifically, forks are created when a significant probability deviation emerges between the initial and current steps. Leveraging this tree structure, ARPO applies GRPO to reinforce LLMs for tool use. In contrast, our proposed PORTool is more efficient in rollout generation and explicitly quantifies each step's contribution to the final outcome. As demonstrated in Section 4.2, ARPO's heavier rollout process and lack of step-wise attribution lead to inferior performance compared to the proposed PORTool method.

Tool-use LLM Training. Beyond RL-based approaches, another widely adopted strategy for equipping LLMs with tool-use capability is supervised fine-tuning (SFT) (Schick et al., 2023; Qin et al., 2023). These methods typically rely on large, high-quality human-annotated datasets such as ToolACE (Liu et al., 2024a), xLAM (Liu et al., 2024b; Prabhakar et al., 2025), and others (Chen et al., 2023, 2024; Acikgoz et al., 2025). However, models trained solely through SFT often face limitations in generalization, exploration, and adaptability (Chu et al., 2025a). In our work, we eliminate human efforts in labeling valid trajectories for all queries and introduce an RL algorithm to encourage LLMs to discover proper tool-call trajectories through trial and error.

3 PORTool: Policy Optimization with Rewarded Tree

A tool-use LLM is equipped with *tool-integrated reasoning* (TIR), in which a query is solved by interleaving tool-configuration reasoning and tool outputs. At each step, the model analyzes the current query in light of previous tool outputs, decides which tools to invoke, and generates the corresponding calls. The returned outputs are incorporated into the model's context, enabling it to refine its reasoning and progress to the next step. In this approach to solving a given query, a sequence of interdependent steps is formed and is known as a *tool-call trajectory*.

In this section, we introduce our design that enables an LLM with the tool-use capability. Therefore, we first formulate an RL objective function that can achieve the goal in Section 3.1. Building on this objective, Section 3.2 introduces PORTool, the proposed method that reinforces an LLM to utilize external tools.

3.1 Problem Formulation

Dataset and Tool set. We consider training an LLM with RL using both a dataset and a tool set:

- **Dataset Q:** The dataset includes a mixture of *time-invariant* queries with fixed factual answers (e.g., "Who is younger, Trump or Biden?") and *time-sensitive* queries requiring real-time information (e.g., "Which of these stocks are above 350 EUR: Apple, Tesla, or Microsoft?"). Notably, some queries admit multiple valid tool-call trajectories. For instance, the stock-price query can be solved either by (i) retrieving the three stock prices in USD and converting the 350 EUR threshold into USD, or (ii) converting the stock prices into EUR and directly comparing them against 350 EUR.
- **Tool set U:** The tool set consists of numerous callable functions, each identified by a unique name and associated with a schema of parameters specifying format and validity constraints. A tool processes the given parameters and returns structured outputs; violations of input requirements result in explicit error messages, while

incorrect configurations would yield unexpected or incomplete results.

Agentic System. The tool-use LLM is optimized within an autonomous agentic system that comprises two distinct roles: a *tool-use agent* responsible for generating tool-call trajectories, while an *evaluation agent* for verifying their correctness.

- **Tool-use Agent π_θ:** This agent loads the LLM to be optimized. Each query is resolved through multiple steps: Starting from an input query, the agent produces a `<think> ... </think>` block containing its reasoning traces, followed by a `<tool_call> ... </tool_call>` block specifying one or more tool calls that can be executed simultaneously and independently. Based on the generated tool calls, the agent invokes relevant tools and obtains their responses, which extend the input context to the LLM. The agent conditions on the new input context to generate another step of reasoning traces and tool calls. This process executes repeatedly until one of the following conditions is met: (i) the query is resolved, signaled by invoking the `response_gen` tool that includes the final answer; or (ii) the query remains unresolved but the maximum number of tool-call steps is reached.
- **Evaluation Agent $R_{out}(q, \tau)$:** This agent uses a more powerful LLM (e.g., GPT) to assess the correctness of the final answer. It generates a high-quality reference answer to the query q through its own reasoning and tool-use capabilities, prioritizing the use of existing knowledge within the trajectory τ. The evaluation returns a binary judgement ("*true*" or "*false*") indicating whether the tool-use agent's final response is correct. In special cases, it may instead output *unable_to_answer* when the tool-use agent provides valid reasoning but cannot reach an answer due to inherent limitations (e.g., tools fail to return the required information but comply with their formatting requirements). This case differs from a wrong answer because the tool-call trajectory would be valid to produce a correct answer when the necessary information is available.

Problem Formulation. Consider sampling a query q from the given dataset Q. The tool-use agent is responsible to generate a tool call trajectory $\tau \triangleq \{q, (s_1, a_1), \ldots, (s_T, a_T)\}$. Specifically, for all $t \in [T]$, $s_t \sim \pi_\theta(s_t|q, U, \{(s_i, a_i)\}_{i<t})$ is the generated content by the LLM for the t-th step, which includes both reasoning traces and tool calls; $a_t = U(s_t)$ is the responses for the tool calls within s_t (or the error messages if there exists any error in s_t). Let us consider that a query q would create n different tool-call trajectories ($n \geq 1$), denoted as $\{\tau_j\}_{j\in[n]}$, where $\tau_j = \{q, (s_{j,1}, a_{j,1}), \ldots, (s_{j,T_j}, a_{j,T_j})\}$. Therefore, we formulate an objective function by maximizing

$$J(\theta) = \mathbb{E}_{\substack{q\sim Q, \\ \{\tau_j\}_{j=1}^n \sim \pi_{\theta_{\text{old}}}(\tau|q,U)}} \Big[\mathcal{L}_{\text{emp}}(\theta; \{\tau_j\})\Big], \quad (1)$$

in which

$$\mathcal{L}_{\text{emp}}(\theta; \{\tau_j\}) \triangleq \frac{1}{n}\sum_{j=1}^{n} \frac{1}{|\tau_j|} \sum_{t=1}^{T_j} \sum_{o=1}^{|s_{j,t}|} f_\theta(s_{j,t,o}),$$

where $|\tau_j|$ indicates the total length of generated content, i.e., $|\tau_j| = \sum_{t\in[T_j]} |s_{j,t}|$; $s_{j,t,o}$ is the o-th token of a tool-call step $s_{j,t}$; and $f_\theta(s_{j,t,o}) \triangleq$

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

where $A(s_{j,t,o})$ indicates the advantage of the token $s_{j,t,o}$, and $\rho_\theta(s_{j,t,o})$ is defined as

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

Limitations of GRPO in tool-use LLM training. Different from the objective formulated by GRPO (Guo et al., 2025), Problem 1 eliminates a KL-divergence term used to align with a reference model. This change provides greater flexibility, enabling the LLM to follow desired output formats and various tool-call schemas through a structured reward mechanism (Qu et al., 2025). In the above objective function, a critical aspect is the estimation of the token-level advantage $A(s_{j,t,o})$, which quantifies the relative contribution of each token to the final outcome. Existing methods such as GRPO (Guo et al., 2025; Shao et al., 2024) simplify this by assigning a uniform advantage across all tokens, thereby neglecting the varying influence of different steps within a trajectory. This limitation primarily arises from the GRPO design where tool-call trajectories are generated independently. As a result, it becomes difficult to determine whether an alternative action at a given step could still recover a correct answer when the trajectory as a whole deviates from the desired outcome.

3.2 Algorithm Description

In this section, we introduce PORTool to overcome the limitations of GRPO by capturing the varying importance of each step during training. As

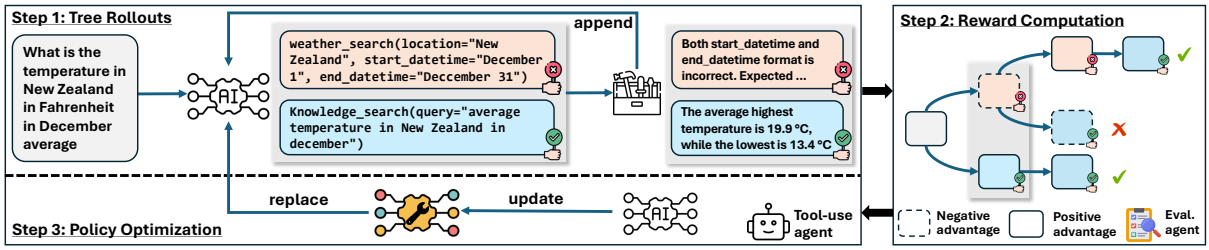


Figure 2: Overview of PORTool Workflow

illustrated in Figure 2, PORTool generates and organizes tool-call trajectories into a tree structure rather than treating them independently. This structure allows rewards to be assigned at each step, reflecting both (i) the correctness of trajectory outcomes and (ii) compliance with formatting requirements and tool-specific schemas. By integrating these step-level rewards, the optimization of the formulated objective (Problem 1) allows the model to gradually equip itself with robust tool-use capabilities. This three-step training procedure is repeated across multiple rounds until convergence.

The rest of the section elaborates on these three steps, including the rollout design, the construction of the reward function, and the optimization strategy for solving Problem 1.

Step 1: Tree Rollouts. Given as input a query q sampled from the dataset Q, the tool-use agent generates n tool-call trajectories, with each fork constrained by a maximum branching factor f, thereby forming a tree structure. The process begins by generating n rollouts for the first tool-call step. At each subsequent step, completed trajectories are collected into a set `completed`, while the remaining incompleted trajectories are stored in `incompleted`. Let $n_c = |\texttt{completed}|$ denote the number of completed trajectories. For each incompleted trajectory, the agent produces f candidate forks and invokes external tools to obtain their responses. From these candidates, exactly $n - n_c$ are selected to continue expansion, while the rest are discarded. The agent terminates the iterative process once either (i) all n trajectories are completed ($n_c = n$) or (ii) the maximum tool-call depth is reached. This guarantees that exactly n trajectories are generated for each query, where some may share common prefixes of tool-call steps before diverging. Algorithm 1 summarizes the rollout process in pseudocode form.

The generated trajectories naturally induce a tree structure for each query. To formalize this, let $m(s_{j,t})$ be a mapping function that returns the set of trajectory indices that pass through the same step $s_{j,t}$. For each step $s_{j,t}$, we define the child set $\mathcal{C}(s_{j,t})$ as the collection of trajectory indices branching from $s_{j,t}$, corresponding to distinct $(t+1)$-th steps. By construction, for any $k_1, k_2 \in \mathcal{C}(s_{j,t})$ with $k_1 \neq k_2$, it holds that $s_{k_1,t+1} \neq s_{k_2,t+1}$. However, it is possible for two distinct trajectories $k_1, k_2 \in m(s_{j,t})$ to satisfy $s_{k_1,t+1} = s_{k_2,t+1}$. Therefore, the relation between two sets is $\mathcal{C}(s_{j,t}) \subseteq m(s_{j,t})$.

Step 2: Reward Computation. With the generated trajectories, we compute rewards from two complementary perspectives: *outcome* R_{out} and *formatting* R_{fm}. For a trajectory τ_j, the reward at its t-th step $s_{j,t}$ is defined as $R(s_{j,t}) \triangleq$

$$G\left(\left\{\gamma^{T_k - t} R_{out}\left(q, \tau_k\right) + R_{fm}\left(s_{j,t}\right)\right\}_{k \in m(s_{j,t})}\right), \tag{2}$$

where $\gamma \in [0, 1]$ is a decay factor ensuring that steps closer to the final result have greater influence, thereby favoring trajectories that solve the task with fewer steps (Feng et al., 2025b). The operator G is chosen adaptively: (i) $G = \max$ if the rewards of all sibling nodes are not identical, i.e., there exists a pair $k_1, k_2 \in \mathcal{C}(s_{j,t-1})$ such that $R(s_{k_1,t}) \neq R(s_{k_2,t})$; and (ii) $G = \text{avg}$ otherwise. This formulation encourages the model to prioritize high-quality tool-call steps that yield higher outcome rewards while maintaining proper formatting. When all sibling steps under a fork achieve equivalent optimal performance, the averaging rule prefers the trajectory with higher probability and efficiency of leading to a correct final answer. Building upon the reward formulation (i.e., Equation 2), we elaborate on the design of outcome and formatting rewards.

- **Outcome reward R_{out}:** As defined in Section 3.1, the outcome reward is determined by the evaluation agent, which labels each trajectory as *true*, *false*, or *unable_to_answer*, corresponding to $\{+1, -1, 0\}$, respectively.

• **Formatting reward R_{fm}:** For each step $s_{j,t}$, we evaluate whether the generated output adheres to the required structural format as described in the system prompt (Appendix D). According to the description of the tool-use agent, the LLM is supposed to first produce a reasoning block, followed by a tool-call block. Thus, we design a concrete rubric for assigning formatting rewards:

1. **Reasoning:** If the generated response includes a `<think> ... </think>` block, the step receives a reward of $+0.2$.

2. **Formatting:** Formatting rewards are activated only when a reasoning block is present. Each tool call must be invoked within a `<tool_call> ... </tool_call>` block and follow a valid JSON format containing a list of tool calls. We assign rewards based on the following criteria:

- If the tool-call indicator `<tool_call> ... </tool_call>` is included, reward $+0.1$.
- If the enclosed content can be parsed as valid JSON, reward an additional $+0.1$.
- If every tool call includes both required fields, namely `"name"` of string type and `"arguments"` of dictionary format, reward an additional $+0.05$.
- Finally, the success of tool invocation is assessed, but only when the above requirements are met. Each tool call is evaluated independently. A maximum reward of $+0.55$ is given if all tool calls succeed, with partial credit assigned in proportion to the fraction of successful tool calls. A failed tool call usually does not comply with the tool-call schemas.

Discussion: Range of formatting rewards. Based on the definitions above, the outcome reward takes values in $\{-1, 0, +1\}$, while the formatting reward lies in the range $[0, 1]$. If these ranges were left unadjusted, undesirable cases could arise where a step with an incorrect outcome but perfect formatting outweighs a step with a correct outcome but imperfect formatting. To prevent this, we rescale the formatting reward so that achieving a correct final answer is always more important than adhering to formatting. Specifically, by assuming the decay factor has an upper bound Γ and the problem must be solved within at most $T_{\max}$ steps, we restrict the formatting reward to $[-\Gamma^{T_{\max}}/2, \Gamma^{T_{\max}}/2]$. In this work, we set $\Gamma = 0.9$ and $T_{\max} = 5$, yielding a rescaled formatting reward range of $[-0.25, 0.25]$. This design ensures that formatting influences the model training but never overrides the impact of correctness.

Step 3: Policy Optimization. In the final stage, we compute token-level advantages to guide the optimization of the policy model (i.e., LLM). Two complementary forms of advantage are considered: (i) a *trajectory-based advantage* $A_{\text{trj}}(\tau_j)$, which evaluates the relative quality of a complete trajectory among the n candidates based solely on the outcome reward; and (ii) a *fork-wise advantage* $A_{\text{fork}}(s_{j,t})$, which evaluates the relative quality of a step compared with its sibling tool-call steps 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 for

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

For all tokens within a step $s_{j,t}$, the advantage is thereby defined as $A(s_{j,t,o}) \triangleq$

$$\frac{\omega_1}{|m(s_{j,t})|} \sum_{k\in m(s_{j,t})} A_{\text{trj}}(\tau_k) + \omega_2 A_{\text{fork}}(s_{j,t}), \quad (3)$$

where ω_1 and ω_2 are weighting coefficients balancing trajectory-level and fork-level influences. This formulation prevents a shared step $s_{j,t}$ that has various advantages across different trajectories. The trajectory-based advantage provides a coarse-grained measure of the overall effectiveness of a tool-call trajectory, offering advantage estimation for every generated token. In contrast, the fork-wise 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 3) 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 for ω_1 and ω_2. The coefficient setting plays a crucial role in training, as it determines the balance between complete trajectory quality and individual tool-call step optimality. One may follow Feng et al. (2025b) to set both to 1. However, since these two terms capture distinct aspects of tool-call generation, equal weighting may introduce **objective inconsistency**, leading to unstable training and suboptimal convergence. To mitigate this issue, we rescale the fork-wise advantage term so that it concentrates on optimizing each tool-call step conditioned on its input context,

without overshadowing the broader trajectory-level objective. A formal definition of the coefficient setting is provided in Theorem 3.1, and Appendix B provides the corresponding analysis.

Theorem 3.1. *By setting $\omega_1 = 1$ and*

$$\omega_2 = \frac{n \times |\tau_j|}{|m(s_{j,t})| \cdot |s_{j,t}| \cdot |\mathcal{C}(s_{j,t-1})| \cdot n_{forks}(q)},$$

where $n_{forks}(q)$ is defined for the size of a set $\{s_{k,t} \mid |\mathcal{C}(s_{k,t})| > 1, k \in [n], t \in [T_k]\}$, which means the number of forks under the tree rollout for the query q. Then, we have the $J(\theta) = J_{GRPO_trj}(\theta) + J_{GRPO_fork}(\theta)$.

4 Experiments

4.1 Setup

Datasets and Models. We construct a dataset of 2,701 complex queries spanning both *time-invariant* and *time-sensitive* cases as mentioned in Section 3.1. Among these, 2,560 queries are used for training and 141 are reserved for evaluation. In this section, we focus on training `Qwen-2.5-7B` (Bai et al., 2023) and `Qwen-3-1.7B` (Yang et al., 2025). The evaluation agent is loaded with GPT-4o (Achiam et al., 2023) to automatically assess trajectory outcomes.

Tools. We implement a suite of 17 tools, including real-time tools and factual tools. A detailed specification of each tool, including its functionality and schema, is provided in Appendix C.1.

Baselines. We compare PORTool against several representative RL algorithms, including GRPO (Guo et al., 2025), DAPO (Yu et al., 2025), and ARPO (Dong et al., 2025). All baseline methods receive an outcome reward assigned to the full trajectory. We further introduce a variant, `GRPO_fm`, which augments GRPO with a formatting reward that averages step-level formatting scores throughout a trajectory to enhance structured generation.

Evaluation Metrics. Model performance is assessed by (i) accuracy judged by the evaluation agent, (ii) average number of tool-call steps per query, and (iii) unanswerable rate, defined as the proportion of queries for which the model fails to produce an answer within five steps. To ensure robustness against evaluation noise and stochastic tool responses, each trained model is executed through *three independent rollouts*, and each rollout result is evaluated *five times* for accuracy estimation, where the accuracy is provided with the error range. The reported results are the mean averaged across all runs.

Method	Acc. (%)	# tool-call steps	Unanswerable Rate (%)
Qwen-2.5-7B	29.79 ± 1.65	4.76	49.83
+ GRPO	48.26 ± 2.37	4.24	31.91
+ GRPO_fm	49.37 ± 1.95	4.37	28.37
+ DAPO	44.88 ± 1.54	4.59	36.88
+ ARPO	46.43 ± 2.01	4.45	31.20
+ PORTool	**63.12** ± 3.15	**3.80**	**15.96**
Qwen-3-1.7B	7.09 ± 0.71	5.78	91.49
+ GRPO	17.06 ± 1.06	5.34	71.56
+ GRPO_fm	20.60± 0.59	5.25	69.47
+ DAPO	13.84±0.83	5.62	82.32
+ ARPO	14.92± 0.71	5.49	78.11
+ PORTool	**31.80** ± 2.72	**4.66**	**61.70**

Table 1: Baseline Comparison

Additional implementation details are provided in Appendix C due to space constraints.

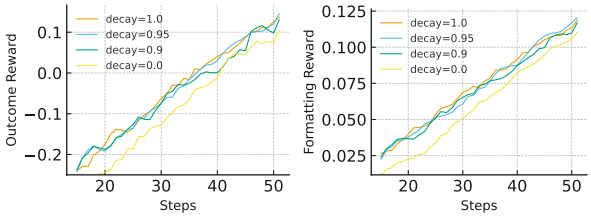
4.2 Comparison with Other Baselines

Table 1 summarizes the performance of the proposed PORTool against a series of RL baselines in terms of evaluation accuracy, average number of tool-call steps, and unanswerable rate.

Compared with the base models (`Qwen-2.5-7B` and `Qwen-3-1.7B`), PORTool exhibits substantial improvements across all metrics. On `Qwen-2.5-7B`, PORTool increases the accuracy from 29.79% to 63.12%, achieving a gain of more than 30%, while reducing the unanswerable rate from 49.83% to 15.96%, a reduction by nearly two-thirds. In addition, the number of required tool-call steps decreases from 4.76 to 3.80, reflecting a more concise and goal-oriented reasoning process. A similar trend is observed for `Qwen-3-1.7B`, where PORTool nearly doubles the accuracy (from 17.06% with GRPO to 31.80%), while cutting the unanswerable rate by almost 10% and reducing tool-use depth.

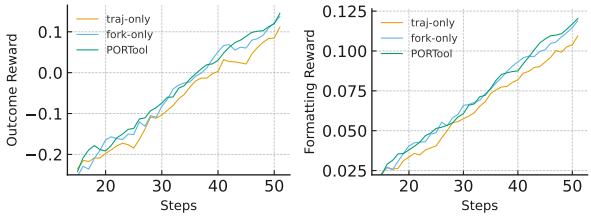
When compared with advanced RL baselines such as GRPO, DAPO, and ARPO, the superiority of PORTool remains consistent. For `Qwen-2.5-7B`, it surpasses the strongest competing method (GRPO_fm) by about 14 percentage points in accuracy, reduces the average tool-call steps by roughly 10%, and decreases the unanswerable rate by over 12 percentage points. The improvements on `Qwen-3-1.7B` show similar magnitudes, confirming that the advantages of PORTool generalize across different backbones.

These results collectively demonstrate that the pro-



(a) Outcome Reward (b) Formatting Reward

Figure 3: Comparison of different decay factors γ.



(a) Outcome Reward (b) Formatting Reward

Figure 4: Comparison of different advantage settings.

posed optimization framework achieves a better balance between accuracy, efficiency, and reliability. By jointly modeling the trajectory-level and fork-wise learning objectives, PORTool encourages the model to perform precise tool invocation while avoiding redundant exploration, resulting in more stable and interpretable reasoning behaviors.

4.3 Ablation Studies

Effectiveness of the decay factor γ. The decay factor γ plays a crucial role in balancing trajectory optimization and step-wise reward refinement. As shown in Figure 3, $\gamma = 0$ results in substantially inferior performance in both outcome and formatting rewards. This behavior arises because $\gamma = 0$ restricts the optimization to step-level formatting correctness, preventing the model from considering the cumulative quality of the trajectory. In contrast, $\gamma = 1.0$ leads the model to focus excessively on trajectory correctness while neglecting reasoning efficiency. Although it improves the outcome reward, this configuration often results in redundant or erroneous tool calls, thereby limiting gains in formatting quality.

Among the tested configurations, $\gamma = 0.95$ achieves the best overall trade-off. It maintains a comparable outcome reward to $\gamma = 1.0$ and yields a superior formatting reward. In contrast, $\gamma = 0.9$ performs suboptimally, marking the boundary where trajectory correctness and formatting correctness exert similar influence. These findings demonstrate that trajectory correctness should be weighted more heavily than formatting accuracy, as emphasizing efficient and accurate trajectories inherently reduces formatting errors and accelerates inference. Therefore, we adopt $\gamma = 0.95$ as the default configuration in all experiments, given its effectiveness in producing concise, accurate, and robust reasoning behaviors.

Different Advantage Settings. As illustrated in Figure 4, the proposed PORTool consistently outperforms both the *trajectory-only* and *fork-only* variants in terms of outcome and formatting rewards. The fork-only configuration achieves stronger results than the trajectory-only variant, as it explicitly captures local step-level consistency and integrates trajectory-level effectiveness into the reward computation. However, its limitation lies in handling steps without sibling nodes, where the fork-wise advantage collapses to zero. This prevents the model from recognizing the importance of such steps in the final reasoning outcome.

In contrast, the trajectory-only variant aggregates rewards over complete trajectories, but it lacks the granularity to distinguish between meaningful and uninformative steps. For instance, an incorrect step within a correct trajectory may still receive positive feedback, while a valuable step in an otherwise incorrect trajectory is penalized. By combining both perspectives, PORTool effectively quantifies the varying contributions of each step and balances trajectory correctness with step formatting correctness. This design enables PORTool to learn concise and accurate reasoning paths, thereby yielding more stable and interpretable tool-use behaviors.

5 Conclusion

In this work, we propose PORTool, a framework for evolving pre-trained LLMs with end-to-end tool-use capabilities under an autonomous agentic setting. Central to PORTool is a tree-rollout strategy that structures tool-call trajectories as a tree, enabling correctness evaluation across multiple branches. Rewards are then assigned to both complete trajectories and intermediate steps, capturing overall effectiveness while distinguishing the most promising decisions within each trajectory. By jointly optimizing trajectory-level and step-wise rewards, PORTool trains LLMs to reason with external tools more robustly. Empirical studies demonstrate that PORTool consistently outperforms existing baselines.

Limitations

One limitation of this work lies in adapting to the continuous evolution of tool schemas and the integration of newly introduced tools, particularly when their interfaces or schemas are not explicitly defined. Such ambiguity may hinder the model's ability to generalize across unseen or inconsistently described tools. To mitigate this challenge, a periodic retraining or fine-tuning process based on the updated tool schemas is necessary to ensure the model remains aligned with the evolving tool ecosystem.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Emre Can Acikgoz, Jeremiah Greer, Akul Datta, Ze Yang, William Zeng, Oussama Elachqar, Emmanouil Koukoumidis, Dilek Hakkani-Tür, and Gokhan Tur. 2025. Can a single model master both multi-turn conversations and tool use? coalm: A unified conversational agentic language model. *arXiv preprint arXiv:2502.08820*.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*.

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

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024. Agent-flan: Designing data and methods of effective agent tuning for large language models. *arXiv preprint arXiv:2403.12881*.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. 2025a. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*.

Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. 2025b. Gpg: A simple and strong reinforcement learning baseline for model reasoning. *arXiv preprint arXiv:2504.02546*.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. 2025. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*.

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

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

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

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

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

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

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

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

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. 2024a. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. 2024b. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu,

Haolin Chen, Thai Hoang, et al. 2025. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*.

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

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

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

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *arXiv preprint*. ArXiv:2305.18290 [cs].

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

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

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. 2025. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *arXiv preprint arXiv:2509.02479*.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. 2025. Demystifying long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2502.03373*.

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

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

Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. 2025. Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning. *arXiv preprint arXiv:2505.00024*.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. 2025. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*.

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. 2025. The surprising effectiveness of negative reinforcement in llm reasoning. *arXiv preprint arXiv:2506.01347*.

The Use of Large Language Models (LLMs)

In this section, we clarify that the term large language model (LLM) refers exclusively to research assistance, including but not limited to idea conception, paper writing, and algorithm implementation. This is distinct from the models employed in our experiments. For manuscript preparation, we used GPT-5 to polish paragraphs by improving clarity and correcting grammatical errors, following a process in which we first drafted the content and then refined it through LLM interaction. After

completing the draft, we conducted comprehensive proofreading and revisions without further LLM assistance. We emphasize that the involvement of LLMs was restricted to writing support; all algorithm design, implementation, and experimental work presented in this paper were carried out solely by the authors.

A Algorithm for Tree Rollout

Algorithm 1 Tree Rollout

Input: Query q, Number of Rollouts n, Maximum number of forks f, Maximum depth $T_{\max}$, Tool-use LLM π_θ, Tool set U

Required: IsCompleted(τ) is a function to justify whether the trajectory τ is completed, i.e., the last step of trajectory τ includes a tool call `response_gen`.

Required: DupAndSamp(`traj`, f, N) duplicates every element in the set `traj` for f times, forming a new set $\overline{\texttt{traj}}$, selects N elements from the new set and returns them.

```
completed ← []
▷ Initialize n trajectories with the first tool-call step
traj ← ∅
for all j ∈ {1, ..., n} do
    s_{j,1} ~ π_θ(s|q)
    a_{j,1} ← U(s_{j,1})
    τ_j ← [q, (s_{j,1}, a_{j,1})]
    traj ← traj ∪ {τ_j}
end for
t ← 1
▷ Iterative expansion with branching factor f while keeping exactly n trajectories
for t ∈ {2, ..., T_max} do
    for τ ∈ traj do
        if IsCompleted(τ) then
            completed.append({τ})
            traj ← traj \ {τ}
        end if
    end for
    n_c ← |completed|
    if n_c = n then
        Break
    end if
    C ← ∅
    traj ← DupAndSamp(traj, f, n − n_c)
    for all τ ∈ traj do
        s_t ~ π_θ(s|q, τ)
        a_t ← U(s_t)
        τ.append((s_t, a_t))
    end for
end for
completed.append(traj)
return completed
```

B Proof of Theorem 3.1

Fork set and notation. Let $\mathcal{C}(s)$ be the index set of children of step s, and $m(s)$ map a step to the index set of its descendant steps whose tokens we average when we expand beyond s (both return sets of indices). Define the set of forked steps

$$\mathcal{F} \triangleq \{\, s_{j,t} \;:\; |\mathcal{C}(s_{j,t})| > 1,\; j \in [n],\; t \in [T_j] \,\},$$

and write $n_{\text{forks}}(q) \triangleq |\mathcal{F}|$ for query q.

Token-level GRPO losses. Recall the objective in (1):

$$J(\theta) = \mathbb{E}\Big[\mathcal{L}_{\text{emp}}(\theta; \{\tau_j\})\Big],$$

$$\mathcal{L}_{\text{emp}}(\theta; \{\tau_j\}) = \frac{1}{n}\sum_{j=1}^{n} \frac{1}{|\tau_j|} \sum_{t=1}^{T_j} \sum_{o=1}^{|s_{j,t}|} f_\theta(s_{j,t,o}),$$

with the standard PPO/GRPO per-token term

$$f_\theta(s) \triangleq \min\Big(\rho_\theta(s)A(s),\, \mathsf{clip}_{1-\varepsilon}^{1+\varepsilon}(\rho_\theta(s))A(s)\Big).$$

We instantiate an *additive* token loss (a standard multi-objective construction) as

$$f_\theta(s) \;=\; f_\theta^{(\text{trj})}(s) \;+\; f_\theta^{(\text{fork})}(s),$$

where

$$f_\theta^{(\text{trj})}(s) \triangleq \min\Big(\rho_\theta(s)A_{\text{trj}}(s),\; \mathsf{clip}_{1-\varepsilon}^{1+\varepsilon}(\rho_\theta(s))A_{\text{trj}}(s)\Big),$$

and, for a *rescaled* fork advantage $\overline{A}_{\text{fork}}$ to be determined,

$$f_\theta^{(\text{fork})}(s) \triangleq \min\Big(\rho_\theta(s)\overline{A}_{\text{fork}}(s),\; \mathsf{clip}_{1-\varepsilon}^{1+\varepsilon}(\rho_\theta(s))\overline{A}_{\text{fork}}(s)\Big).$$

By construction,

$$J(\theta) = J_{\text{GRPO_trj}}(\theta) + J_{\text{GRPO_fork}}(\theta),$$

with $J_{\text{GRPO_trj}}$ and $J_{\text{GRPO_fork}}$ the expectations of the empirical sums formed with $f_\theta^{(\text{trj})}$ and $f_\theta^{(\text{fork})}$ respectively. We set $\omega_1 = 1$ in A_{trj}.

Fork-wise empirical objective. Independently, define the fork-wise objective by averaging over forks, then over children and their tokens:

$$\mathcal{L}_{\text{emp}}^{(\text{fork})}(\theta; \{\tau_j\}) \triangleq \frac{1}{|\mathcal{F}|} \sum_{s_{j,t} \in \mathcal{F}} \mathcal{L}_{\text{fork}}(\theta \mid s_{j,t}),$$

where

$$\mathcal{L}_{\text{fork}}(\theta \mid s_{j,t}) = \frac{1}{|\mathcal{C}(s_{j,t})|} \sum_{k \in \mathcal{C}(s_{j,t})} \frac{1}{|s_{k,t+1}|} \sum_{o=1}^{|s_{k,t+1}|} f_\theta^{(\text{fork})}(s_{k,t+1,o})$$

Then $J_{\text{GRPO_fork}}(\theta) \triangleq \mathbb{E}[\mathcal{L}_{\text{emp}}^{(\text{fork})}(\theta; \{\tau_j\})]$.

Reindexing lemma (disjoint union). For any fork $s_{j,t} \in \mathcal{F}$,

$$m(s_{j,t}) = \bigcup_{k \in \mathcal{C}(s_{j,t})} m(s_{k,t+1}),$$

$$m(s_{k,t+1}) \cap m(s_{k',t+1}) = \emptyset \;\text{ for } k \neq k'.$$

Consequently,

$$\sum_{k \in \mathcal{C}(s_{j,t})} \frac{1}{|s_{k,t+1}|} \sum_{o=1}^{|s_{k,t+1}|} f_\theta^{(\text{fork})}(s_{k,t+1,o})$$

$$= \sum_{k \in m(s_{j,t})} \sum_{o=1}^{|s_{k,t+1}|} \frac{f_\theta^{(\text{fork})}(s_{k,t+1,o})}{|m(s_{k,t+1})|\,|s_{k,t+1}|},$$

This is exactly the equalities used informally in your draft, now stated as a lemma.

Matching normalizers and the choice of ω_2. Insert the lemma into $\mathcal{L}_{\text{emp}}^{(\text{fork})}$:

$$\mathcal{L}_{\text{emp}}^{(\text{fork})}(\theta; \{\tau_j\})$$

$$= \frac{1}{|\mathcal{F}|} \sum_{s_{j,t} \in \mathcal{F}} \sum_{k \in m(s_{j,t})} \sum_{o=1}^{|s_{k,t+1}|} \frac{f_\theta^{(\text{fork})}(s_{k,t+1,o})}{|\mathcal{C}(s_{j,t})|\,|m(s_{k,t+1})|\,|s_{k,t+1}|}.$$

To embed this into the per-trajectory/token averaging of (1), observe that a token $s_{u,o}$ appearing as a child of a forked *parent* step $s_{j,t-1}$ receives weight

$$\underbrace{\frac{1}{|\mathcal{F}|}}_{\text{fork avg}} \cdot \underbrace{\frac{1}{|\mathcal{C}(s_{j,t-1})|}}_{\text{children avg}} \cdot \underbrace{\frac{1}{|m(s_{j,t})|}}_{\text{descendants split}} \cdot \underbrace{\frac{1}{|s_{j,t}|}}_{\text{token avg}}.$$

In contrast, in (1) the empirical averaging for any token of trajectory j is $\frac{1}{n} \cdot \frac{1}{|\tau_j|}$. Therefore, to write the fork contribution *in the same empirical form as* (1), we absorb the ratio of these weights into the advantage via

$$\overline{A}_{\text{fork}}(s_{j,t,o}) \triangleq \omega_2\, A_{\text{fork}}(s_{j,t,o}),$$

where

$$\omega_2 \;=\; \frac{n\,|\tau_j|}{|m(s_{j,t})|\,|s_{j,t}|\,|\mathcal{C}(s_{j,t-1})|\,n_{\text{forks}}(q)}.$$

(When $s_{j,t}$ is not a child of a fork, take $\overline{A}_{\text{fork}}(s_{j,t,o}) = 0$.) With this choice,

$$\mathcal{L}_{\text{emp}}^{(\text{fork})}(\theta; \{\tau_j\}) = \frac{1}{n}\sum_{j=1}^{n}\frac{1}{|\tau_j|}\sum_{t=1}^{T_j}\sum_{o=1}^{|s_{j,t}|} f_\theta^{(\text{fork})}(s_{j,t,o}).$$

Combining with the trajectory term (where we set $\omega_1 = 1$),

$$\mathcal{L}_{\text{emp}}(\theta; \{\tau_j\}) = \frac{1}{n}\sum_{j=1}^{n}\frac{1}{|\tau_j|}\sum_{t,o}\Big(f_\theta^{(\text{trj})}(s_{j,t,o}) + f_\theta^{(\text{fork})}(s_{j,t,o})\Big),$$

and taking expectations yields

$$J(\theta) = J_{\text{GRPO_trj}}(\theta) + J_{\text{GRPO_fork}}(\theta).$$

Remarks. (i) The additivity $f_\theta = f_\theta^{(\text{trj})} + f_\theta^{(\text{fork})}$ is the usual way to combine multiple GRPO/PPO-style objectives; we do *not* replace A by a single sum inside one $\min(\cdot)$ (which would not in general be additive). (ii) The scaling ω_2 is the unique choice (up to a constant that cancels if applied to both numerator and denominator) that aligns the fork averaging with the per-trajectory/token averaging in (1).

C Experimental Setup

C.1 Tool Descriptions

In this work, we integrate a set of structured function-based tools to support experiments involving real-time information retrieval, knowledge reasoning, and contextual decision-making. Each tool is defined with a specific schema to ensure precise query formulation and reliable execution. Below, we describe the functionalities of the tools used:

- **News Search (`news_search`)**: Provides access to news headlines and articles related to current events, given a query string.
- **Weather Search (`weather_search`)**: Returns current weather conditions and forecasts for a specified location. The tool supports time-bounded queries with start and end times in ISO 8601 format, and allows adjustable precision (e.g., hour, day).
- **Flights Search (`flights_search`)**: Retrieves airline flight information, including schedules, status, arrival, and departure data. It supports flexible specifications such as carrier codes, flight numbers, departure/arrival locations, and temporal windows.
- **Stocks Search (`stocks_search`)**: Fetches real-time and historical stock price information using stock or market names as input queries.
- **Sports Search (`sports_search`)**: Provides information on sports teams, athletes, schedules, and standings based on query intent.
- **Maps Search (`maps_search`)**: Identifies businesses, landmarks, and addresses, and computes estimated travel times. It allows specification of anchor locations, estimated time of arrival (ETA) type, and ETA constraints.
- **Conversion Calculation (`conversion_calculation`)**: Handles measurement and currency unit conversions. The tool accepts structured inputs specifying the original units and values, the target units, and optionally, the substance to support weight–volume conversions.
- **Math Calculation (`math_calculation`)**: Executes mathematical computations directly based on a textual math query.
- **Get Current Context (`get_current_context`)**: Retrieves contextual information such as the current time and/or the user's current location.
- **Timestamp Comparator (`timestamp_comparator`)**: Compares two ISO 8601 timestamps using operators such as $<$, $>$, or $==$. Time zone information is required for both timestamps.
- **Sort List (`sort_list`)**: Sorts a list of objects by a specified numeric property, in either ascending or descending order.
- **Holiday Search (`holiday_search`)**: Returns holiday information for a specified date, period, or country. It supports filtering by holiday type (e.g., public, bank, cultural).
- **Knowledge Search (`knowledge_search`)**: Retrieves factual knowledge from a structured knowledge graph about entities such as people, places, or organizations. Queries can be refined using entity IDs or labels.
- **Filter List (`filter_list`)**: Filters a list of objects by applying a condition (e.g., $==$, $>$, $<$) to a specified property field.
- **Timestamp Converter (`timestamp_converter`)**: Converts a timestamp

from one time zone to another, supporting both canonical time zone identifiers and location-based descriptions.

- **Timestamp Interval Calculator (`timestamp_interval_calculator`)**: Computes a new timestamp by adding or subtracting an interval (in ISO 8601 duration format) from a given reference timestamp, with explicit control over the operation type (add/subtract).

These tools provide a modular framework for handling diverse queries spanning news, weather, transportation, finance, sports, temporal reasoning, and general knowledge. Their structured input–output schemas enable seamless integration with learning agents, ensuring consistency and reliability in multi-step reasoning and tool-use experiments.

C.2 Implementation

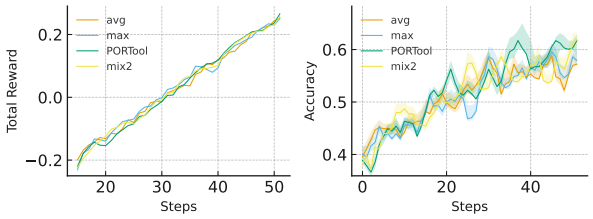
Our implementation is built upon the VeRL framework (Sheng et al., 2024), with rollouts executed through VLLM (Kwon et al., 2023). All experiments are conducted on a single node equipped with eight H100 GPUs. The training batch size is set to 512, and in each update round, 128 samples are used to optimize the policy model. During training, text generation adopts temperature $= 1.0$, top-p $= 1.0$, and top-k $= 0$, while validation employs deterministic decoding (temperature $= 0.0$). Each reasoning step is limited to 2,048 tokens, and for every query, 8 rollouts are generated. For time-sensitive queries, the reference location is fixed to Cupertino, California (USA), and execution timestamps are logged with second-level precision to ensure reproducibility. Prior to reinforcement learning, supervised fine-tuning (SFT) is performed for five epochs. All baselines are trained for 15 epochs (equivalent to 75 steps in total), and we report their best performance across training checkpoints. Unless otherwise specified, we adopt LoRA fine-tuning with rank $= 16$ and $\alpha = 8$. Detailed training and rollout configurations are summarized in Table 2.

C.3 More Experiments

Designs of reward function $G(\cdot)$. As shown in Figure 5, configurations where $G(\cdot)$ is implemented as either a max or avg function perform noticeably worse than the proposed mixed formulation and PORTool. The max-based design tends to overemphasize the best-performing trajectory at each step, causing the model to become stuck when multiple tool-call branches yield identical rewards. In contrast, the avg design exhibits rapid improvement in the early stages of training but plateaus prematurely as training progresses.

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 (10 for ablation studies)

Table 2: Training and rollout hyperparameter configuration.



(a) Total Reward (b) Accuracy

Figure 5: Comparison of different designs of $G(\cdot)$.

To further analyze this, we introduce a variant termed `mix2`, which adopts a distinct formulation from the proposed PORTool, i.e., $G\left(\left\{\gamma^{T_k-t} R_{out}\left(q, \tau_k\right)\right\}_{k \in m(s_{j,t})}\right) + R_{fm}\left(s_{j,t}\right)$. Here, when multiple steps achieve the same optimal outcome reward, `mix2` averages their reward quality, whereas PORTool explicitly integrates the formatting reward into the optimization objective. Experimental results demonstrate that PORTool eventually outperforms `mix2`, suggesting that lower formatting rewards can implicitly constrain the upper bound of trajectory-based rewards.

D System Prompt

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. 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

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: `get_current_context`

Description: Retrieves specific aspects of the user's request context, such as current location and current time.
Parameters:
- **requested_context** (array of strings) A list of context fields to retrieve, encompassing `"current_location"` and/or `"current_time"` only.

Required: { `requested_context` }
Type: object

Tool Schema: `maps_search`

Description. finds businesses, landmarks, and addresses and calculates travel times

Parameters.
- **query** (string). User's search intent. The query should follow one of these patterns:
  1. {navigate me to/find/call/phone number/opening hours/how far/eta}{poi/category/menu items/address}{near/on/around/in/from/to [anchor_location]}
  2. find {poi/category} near [anchor_location]
  3. {how far/eta} from [anchor_location] to [targeted_location]
  4. is [anchor_location] open {now/at time/on day}

  All [anchor_location] and [targeted_location] must be concrete and unambiguous. For queries based on the user's current location, [anchor_location] or [targeted_location] in the above patterns may be filled with "near me", "from here", or "from my location", and *do not* include the *anchor_location* parameter. If a query contains a vague location (e.g., "McDonald near me"), it is forbidden. Instead, first issue "find [location]" to resolve candidates, then always select the first candidate and rewrite the query accordingly (e.g., "is McDonald at 12th Ave open now", "how far is McDonald at 12th Ave from here"). Do not output vague queries like "is McDonald near me open now" or "how far is McDonald".
- **anchor_location** (string). Reference text describing the area or place mentioned in the *query* (e.g., "4th Street", "Central Park"). Leave this empty if the *query* is related to the user's current location.
- **eta_type** (string). Type of ETA query. Allowed values include `"departure"` and `"arrive"`.
- **eta_time** (string). Time constraint associated with *eta_type*. Use ISO 8601 format.

Required. { query }

Type. object

Tool Schema: `knowledge_search`

Description. Returns passages retrieved from the web and the knowledge graph. The tool retrieves top dynamically ranked passages that answer the query. In addition to decomposed queries, it can also address complex queries by sourcing multiple passages. Note: Web passages are less effective at providing real-time and location-sensitive answers.

Parameters.
- **query** (string). The natural language query to search on the web.
- **entity_labels** (array of strings). List of entity labels used to retrieve relevant facts from the knowledge graph.

Required. { query }

Type. object

Tool Schema: `math_calculation`

Description. Performs mathematical calculations using function-call syntax. **Important:** Use only raw numbers in expressions (no currency symbols, units, or % signs). The tool returns numeric results without formatting—add appropriate context (currency, units, etc.) in your response based on the user's question.

Parameters.
- **expression** (string). Mathematical expression using function-call syntax.
  Available functions include: `add(a, b, ...)`, `subtract(a, b)`, `multiply(a, b, ...)`, `divide(a, b)`, `avg(a, b, ...)`, `modulus(a, b)`, `isDivisible(a, b)`, `power(base, exponent)`, `root(number, degree)`, `factorial(n)`, `gcd(a, b, ...)`, `lcm(a, b, ...)`, `factor(n)`, `primeFactor(n)`, `random(min, max)`, `log(number, base)`, `percentOf(value, percent)`, `percentDiscount(value, percent discount)`, `toPercent(decimal)`, `fromPercent(percent)`, `decimalToFraction(decimal)`, `fractionToDecimal(fraction)`.
  Constants: `pi`, `e`, `phi`, `tau`. Nested functions may be used for complex expressions.

Required. { expression }

Type. object

Tool Schema: `conversion_calculation`

Description. Measurement and currency unit conversions.

Parameters.
- **query** (string). The query that contains the conversion to perform.
- **substance** (string). The substance the user wants to include as part of the conversion is used to facilitate weight-to-volume and volume-to-weight conversions.
- **from** (array of objects). The units and their respective amounts to convert. Each object must include:
  - `unit` (string): The unit to convert.
  - `value` (number): The amount or value of the unit.
- **to_units** (array of strings). The units the user wants to convert to. This array must contain only one element, representing the single target unit for the conversion.

Required. { query }

Type. object

Tool Schema: `timestamp_interval_calculator`

Description. Calculates a new ISO 8601 timestamp by adding or subtracting a time interval from a given ISO 8601 timestamp.

Parameters.
- **from_timezone** (string). The time zone corresponding to the `from_timestamp`.
- **from_location** (string). The location corresponding to the `from_timestamp`.
- **from_timestamp** (string). Input timestamp in ISO 8601 format (`YYYY-MM-DDThh:mm:ss`). Do not include the time zone offset, as it is assumed to be in `from_timezone`.
- **interval** (string). Time interval to be added or subtracted, in ISO 8601 duration format (`P[n]Y[n]M[n]DT[n]H[n]M[n]S`).
- **operation** (string). The operation to perform. Allowed values: `add`, `subtract`.

Required. { from_timestamp, interval, operation }

Type. object

Tool Schema: `response_gen`

Description. Response generation.

Parameters.
- **response** (string). Response to be generated.
- **query** (string). User query or rewritten query.
- **context** (string). Context for response generation.
- **items** (array of objects). Items with entities and facts.

Required. { response }

Type. object

E Example I: which team has better performance in the last three seasons: the steelers, ravens, or eagles?

Method	Generated Result
GRPO	I encountered an issue retrieving the specific performance data for the Steelers, Ravens, and Eagles for the last three seasons. Please provide more specific details or context for the sport and league you are interested in.
ARPO	There seems to be an issue with retrieving the football standings for the Steelers, Ravens, and Eagles. Please try again later or use a different source for this information.
DAPO	The performance data for the Steelers, Ravens, and Eagles in the last three seasons is not available in the expected format. Based on the information provided, we cannot determine which team has the better performance.
PORTool	Based on the data extracted, the Raptors (Ravens) had a positive performance in the last three seasons. They won 12 out of their final 15 games and finished the season with a record of 12-5. On the other hand, the Steelers had a record of 4-2 after snapping their losing streak in Week 6. The Eagles, however, faced an embarrassing collapse, losing five out of their last six games. Therefore, in the last three seasons, the Ravens had a better performance compared to the Steelers and Eagles.

Table 3: Generated results across different methods with Qwen-2.5-7B

E.1 PORTool

Query: *Which team has better performance in the last three seasons: the Steelers, Ravens, or Eagles?*

Step 1. To determine which team has better performance in the last three seasons, I need to retrieve the relevant team standings for the Steelers, Ravens, and Eagles.
Tool Call: `{"name": "sports_search", "arguments": {"query": "Steelers standings last 3 seasons", "use_case": {"standings": {"requested_info": "PERFORMANCE"}}}}`
Tool Response:
`{"result": "No data is available"}`
Tool Call: `{"name": "sports_search", "arguments": {"query": "Ravens standings last 3 seasons", "use_case": {"standings": {"requested_info": "PERFORMANCE"}}}}`
Tool Response:
`{"result": "No data is available"}`
Tool Call: `{"name": "sports_search", "arguments": {"query": "Eagles standings last 3 seasons", "use_case": {"standings": {"requested_info": "PERFORMANCE"}}}}`
Tool Response:
`{"result": "No data is available"}`

Step 2. The responses from the sports_search tool are not providing the necessary team performance data. I will try using the knowledge_search tool to find the performance data of the Steelers, Ravens, and Eagles in the last three seasons.

Tool Call: `{"name": "knowledge_search", "arguments": {"query": "Steelers performance last 3 seasons", "entity_labels": ["Steelers"]}}`

Tool Response:

```
[1] Title: 2024 Steelers Season Recall: Losing streak snapped against Las Vegas ...
[2] Title: The latest steelers' offensive struggles surface news, videos & comments | NewsBreak ...
```

Tool Call: `{"name": "knowledge_search", "arguments": {"query": "Ravens performance last 3 seasons", "entity_labels": ["Ravens"]}}`

Tool Response:

```
[1] Title: 2024 Baltimore Ravens season ...
[2] Title: Baltimore Ravens Playoff History | 1996 - 2025 ...
```

Tool Call: `{"name": "knowledge_search", "arguments": {"query": "Eagles performance last 3 seasons", "entity_labels": ["Eagles"]}}`

Tool Response:

```
[1] Title: Eagles' collapse to end 2023 season was a "total misrepresentation" of team according to Lane Johnson | Sporting News ...
[2] Title: Eagles' SOS: AFL's perennial strugglers granted assistance package in revival bid ...
```

Step 3. The knowledge_search tool responses provide information about the performance of each team in the last three seasons. Based on the provided data, I can now determine which team has performed better.

Tool Call: `{"name": "response_gen", "arguments": {"response": "Based on the data extracted, the Raptors (Ravens) had a positive performance in the last three seasons. They won 12 out of their final 15 games and finished the season with a record of 12-5. On the other hand, the Steelers had a record of 4-2 after snapping their losing streak in Week 6. The Eagles, however, faced an embarrassing collapse, losing five out of their last six games. Therefore, in the last three seasons, the Ravens had a better performance compared to the Steelers and Eagles."}}`

Figure 6: Step-by-step reasoning and tool usage under PORTool. Tool calls are shown in blue and tool responses in green. Formatting edition for presentation clarification.

F Example II: What's 70 days from march 21?

F.1 ToolRL

Query: *what's 70 days from march 21*

Step 1. To find the date 70 days from March 21, I need to add 70 days to March 21.
Tool Call: `{"name": "timestamp_interval_calculator", "arguments": {"from_timestamp": "2023-03-21", "interval": "P70D", "operation": "add"}}`
Tool Response:
`ERROR: invalid timestamp: {2023-03-21}. supported time formats include ISO 8601 date-time with or without timezone (e.g., 2006-01-02T15:04:05Z07:00, 2006-01-02 15:04:05), 12-hour AM/PM with or without seconds (e.g., 03:04:05 PM, 03:04PM), and 24-hour formats (e.g., 15:04:05, 15:04)`

Step 2. The timestamp calculator tool returned an error, possibly due to the timestamp format being recognized as incorrect. However, I can directly calculate the date 70 days from March 21 using a simple addition.
Tool Call: `{"name": "timestamp_interval_calculator", "arguments": {"from_timestamp": "2023-03-21", "interval": "P70D", "operation": "add"}}`
Tool Response:
`ERROR: invalid timestamp: {2023-03-21}. supported time formats include ISO 8601 date-time with or without timezone (e.g., 2006-01-02T15:04:05Z07:00, 2006-01-02 15:04:05), 12-hour AM/PM with or without seconds (e.g., 03:04:05 PM, 03:04PM), and 24-hour formats (e.g., 15:04:05, 15:04)`

Step 3. Despite the error, the date can be calculated manually since the tool uses the current year, and the timestamp format is valid for the task. 70 days from March 21 is April 20.
Tool Call: `{"name": "response_gen", "arguments": {"response": "70 days from March 21 is April 20."}}`

Figure 7: Step-by-step reasoning and tool usage under ToolRL. **The generated answer is incorrect.**

F.2 PORTool

Query: *what's 70 days from march 21*

Step 1. To find the date 70 days from March 21, I need to add 70 days to March 21.
Tool Call: `{"name": "timestamp_interval_calculator", "arguments": {"from_timestamp": "2023-03-21", "interval": "P70D", "operation": "add", "from_timezone": "", "from_location": ""}}`
Tool Response:
`ERROR: invalid timestamp: {2023-03-21}. supported time formats include ISO 8601 date-time with or without timezone (e.g., 2006-01-02T15:04:05Z07:00, 2006-01-02 15:04:05), 12-hour AM/PM with or without seconds (e.g., 03:04:05 PM, 03:04PM), and 24-hour formats (e.g., 15:04:05, 15:04)`

Step 2. It seems there was an error with the timestamp format. I will retry using the correct format.
Tool Call: {"name": "timestamp_interval_calculator", "arguments": {"from_timestamp": "2023-03-21T00:00:00", "interval": "P70D", "operation": "add", "from_timezone": "", "from_location": ""}}
Tool Response:
{"calculatedTime": "Tuesday 2023-05-30T00:00:00-07:00 Week_number 22 Day_number 150", "timezoneLocal": "America/Los_Angeles"}

Step 3. The date 70 days from March 21, 2023 is May 30, 2023.
Tool Call: {"name": "response_gen", "arguments": {"response": "The date 70 days from March 21, 2023 is May 30, 2023."}}

Figure 8: Step-by-step reasoning and tool usage under PORTool with Qwen-2.5-7B backbone. **While the method generates a correct answer, it also adds a meaningless year, 2023.** From the evaluation agent's perspective, this answer is labeled as **true**.

G Example III: How many hours to tomorrow

G.1 PORTool with Qwen-2.5-7B

Query: *how many hours to tomorrow*

Step 1. To determine how many hours are until tomorrow, I need to calculate the time difference between now and the start of tomorrow.
Tool Call: `{"name": "timestamp_comparator", "arguments": {"timestamp1": "now", "timezone1": "UTC", "timestamp2": "tomorrow", "timezone2": "UTC", "comparison_operator": ">"}}`
Tool Response:
`ERROR: invalid format for timestamp1: The value must be in a valid ISO 8601 format. Valid templates include: YYYY-MM-DD for date, YYYY-MM-DDThh:mm:ss for local time without timezone, YYYY-MM-DDThh:mm:ssZ for UTC time, and YYYY-MM-DDThh:mm:ss+hh:mm / YYYY-MM-DDThh:mm:ss-hh:mm for times with timezone offsets (ahead or behind UTC time).`

Step 2. It seems there was an error because the 'now' value was not provided in the correct ISO 8601 format. I need the current time in ISO 8601 format to make the comparison.
Tool Call: `{"name": "get_current_context", "arguments": {"requested_context": ["current_time"]}}`
Tool Response:
`{"current_time": {"date_time": "Saturday 2025-09-27T02:47:20-07:00 Week_number 39 Day_number 270", "time_zone": "America/Los_Angeles"}}`

Step 3. `<think> I have the current time in ISO 8601 format now. I can calculate the time difference between now and the start of tomorrow. </think> :UIControlStateNormal [{"name": "timestamp_interval_calculator", "arguments": {"from_timezone": "America/Los_Angeles", "from_location": "Los Angeles", "from_timestamp": "2025-09-27T02:47:20-07:00", "interval": "P1D", "operation": "add"}}] </tool_call>`
Tool Response:
`ERROR: Invalid output format. A valid output must include both: (i) the <think> ... </think> field; and (ii) the <tool_call> ... </tool_call> field.`

Step 4. Oops, it seems there was an error with the tool execution. Let me try calculating the hours to tomorrow directly using the correct format.
Tool Call: `{"name": "timestamp_comparator", "arguments": {"timestamp1": "2025-09-27T02:47:20-07:00", "timezone1": "America/Los_Angeles", "timestamp2": "2025-09-28T00:00:00-07:00", "timezone2": "America/Los_Angeles", "comparison_operator": "<"}}`
Tool Response:
`{"comparisonResult": true, "timeDifference": "21 hours, 12 minutes, 40 seconds"}`

Step 5. There are 21 hours and 12 minutes and 40 seconds until tomorrow.
Tool Call: `{"name": "response_gen", "arguments": {"response": "There are 21 hours and 12 minutes and 40 seconds until tomorrow."}}`

Figure 9: Step-by-step reasoning and tool usage under PORTool with the Qwen-2.5-7B backbone.

G.2 PORTool with Qwen-3-1.7B

Query: *how many hours until tomorrow*

Step 1. Okay, the user is asking, "how many hours until tomorrow." Let me think about how to approach this.
First, I need to determine the current time to calculate the time until midnight. The user mentioned "tomorrow," so I should check the current date and time. Since the user's location isn't specified, I'll assume they're in the same time zone as the system.
Next, I need to figure out the time difference between now and midnight of the next day. If it's already past midnight, the answer would be 0 hours. If it's before midnight, subtract the current time from 24 hours.
Wait, the tools available include weather_search, which can provide current weather conditions. But the user's question is about time, not weather. However, the weather_search tool might have a parameter for start_datetime or end_datetime. But the user isn't asking for weather data; they want the time until tomorrow.
Hmm, maybe the user just wants the time until midnight tomorrow. Since the exact time isn't provided, I need to assume the current time. But the tools don't have a function to get the current time directly. The get_current_context tool can retrieve current location and time.
Wait, the get_current_context function can retrieve current time. So I should use that to get the current time. Then calculate the time until midnight tomorrow.
Let me outline the steps: 1. Use get_current_context to get the current time. 2. Calculate the time until midnight of the next day. 3. Return the result.
But the user's question is straightforward. If the current time is, say, 10:00 AM, then the time until tomorrow's midnight would be 14 hours. If it's 11:00 PM today, then 13 hours until midnight tomorrow.
But since the exact current time isn't provided, I need to use the tool to get the current time. Once I have that, I can compute the difference.
So the correct approach is to call get_current_context to get the current time, then calculate the hours until tomorrow's midnight. The tool will return the current time, and I can use that to determine the answer.
Tool Call: `{"name": "get_current_context", "arguments": {"requested_context": ["current_time"]}}`
Tool Response:
`{"current_time": {"date_time": "Sunday 2025-10-05T16:52:30-07:00 Week_number 40 Day_number 278", "time_zone": "America/Los_Angeles"}}`

Step 2. Okay, the user asked, "how many hours until tomorrow." I need to figure out the time difference between now and midnight of the next day.
First, I used the get_current_context tool to get the current time. The response shows the current time is Sunday, October 5, 2025, at 16:52:30 in Los Angeles. The time zone is America/Los_Angeles.
Now, I need to calculate the time until midnight of tomorrow. Since today is Sunday, tomorrow is Monday. The current time is 4:52 PM. Midnight tomorrow would be 12:00 AM.
The time difference is 8 hours (from 4:52 PM to 12:00 AM is 8 hours). But wait, the current time is 4:52 PM, so from 4:52 PM to 12:00 AM is 7 hours and 8 minutes. Wait, let me check again. From 4:52 PM to 12:00 AM is 7 hours and 8 minutes. But the user might want the exact number of hours. So 7 hours and 8 minutes. But the tool might not have the exact time, so I need to calculate it based on the current time.

Step 2 (cont.). Wait, the current time is 4:52 PM. The time until midnight is 8 hours and 8 minutes. Wait, 4:52 PM to 12:00 AM is 7 hours and 8 minutes. Wait, no. From 4:52 PM to 12:00 AM is 7 hours and 8 minutes. But the exact calculation would be 8 hours if we consider the current time is 4:52 PM, then 12:00 AM is 7 hours and 8 minutes later. So the answer is 7 hours and 8 minutes. But the user might want the answer in hours, so 7.1333 hours, but the tool might not have the exact time. Alternatively, maybe the tool can calculate it, but since the tool only returns the current time, I need to compute it manually.
But since the user's question is straightforward, and the tool gives the current time, I can compute the hours until midnight tomorrow. The current time is 4:52 PM, so 8 hours until midnight. Wait, 4:52 PM to 12:00 AM is 7 hours and 8 minutes. So the answer is 7 hours and 8 minutes. But the user might want it in hours, so 7.1333 hours. However, the exact answer depends on the current time. Since the current time is 4:52 PM, the time until midnight is 7 hours and 8 minutes. So the answer is 7 hours and 8 minutes. But the user might prefer it in hours as a decimal or rounded.
Tool Call: `{"name": "response_gen", "arguments": {"response": "There are 7 hours and 8 minutes until midnight tomorrow."}}`

Figure 10: Step-by-step reasoning and tool usage under PORTool with the Qwen-3-1.7B backbone.

H Example IV: if you're nine years old and in 2030 how old will you be

Method	Generated Result
GRPO	If you are currently 9 years old in 2023, you will be 16 years old in 2030.
ARPO	NA
DAPO	NA
PORTool	If you're nine years old in 2025, you will be 14 years old in 2030.

Table 4: Generated results across different methods with Qwen-3-1.7B

H.1 GRPO

Query: *if you're nine years old and in 2030 how old will you be*

Step 1. Okay, the user is asking, "if you're nine years old and in 2030 how old will you be?" Let me break this down.
First, the user is probably trying to calculate their age in 2030 based on their current age. They mentioned they're nine years old now. So, if today is 2023, then in 2030, they would be 9 + 7 = 16 years old. But the user hasn't specified the current year, so I need to consider the current date.
Wait, but the tools available don't include a way to calculate age based on current date. The tools like weather_search, stocks_search, etc., are specific to their functions. Since there's no tool for date calculations, I can't use any of the provided functions here. The user's question is a simple arithmetic problem that doesn't require any of the tools. So, I should answer directly without using any tools. But according to the instructions, I must use the tools if possible. However, since the question is straightforward and doesn't require external data, I can't invoke any tools. Therefore, I need to respond with the answer based on the current date, but since the current date isn't provided, I can't calculate it precisely. But the user might assume the current year is 2023, so 2023 + 7 = 2030, making the age 16. However, without knowing the exact current date, this is an assumption. But the tools don't support this, so the answer must be given directly.
Tool Call: `{"name": "response_gen", "arguments": {"response": "If you are currently 9 years old in 2023, you will be 16 years old in 2030."}}`

Figure 11: Step-by-step reasoning and tool usage under GRPO. **The generated answer is incorrect because it identifies the current year as 2023.**

H.2 PORTool

Query: *if you're nine years old and in 2030 how old will you be*

Step 1. Okay, the user is asking, "if you're nine years old and in 2030 how old will you be?" Let me break this down.
First, the user is probably asking about their own age in 2030, but they mentioned "you" which might be a typo. But assuming they mean their age, the answer is straightforward. If they're nine now, in 2030 they'll be nine plus the number of years until 2030. But wait, the current year is 2023, right? So from 2023 to 2030 is 7 years. So 9 + 7 = 16. But the user might not know the current year. However, the tools available don't include a way to calculate age based on current date. The tools like math_calculation can handle arithmetic, but the user's question is a simple addition. Since the current year isn't provided, I can't calculate it accurately. But the user might expect a simple answer. However, the tools require specific parameters. Since the user's question is a straightforward math problem, I can use math_calculation with the expression "9 + 7" if the current year is 2023. But the tool needs the current year. Since I don't have that, maybe I should inform the user that the current year is 2023 and the answer is 16. But the tools don't have a way to get the current year. Wait, the get_current_context tool can retrieve current location and time. If I call that, I can get the current year. Let me check the available tools again. The get_current_context tool can retrieve current_location and current_time. So, I should use that to find the current year. Then calculate 2023 + 7 = 2030, so 9 + 7 = 16. But the user might not know the current year. However, the tools can't provide that info unless the user's location is known. Since the user hasn't provided their location, I can't get the current year. Therefore, I need to inform the user that the current year is 2023 and the answer is 16. But the tools don't have a way to get the current year unless the user's location is known. So, the correct approach is to use get_current_context to find the current year, then compute the age. But since the user's question is a simple math problem, maybe the answer is 16, but I need to use the tools to get the current year first. However, the tools might not have the current year. So, I'll proceed to use get_current_context to find the current year, then calculate the age.
Tool Call: `{"current_time": {"date_time": "Sunday 2025-10-05T18:28:16-07:00 Week_number 40 Day_number 278", "time_zone": "America/Los_Angeles"}}`
Tool Response:
`ERROR: invalid timestamp: {2023-03-21}. supported time formats include ISO 8601 date-time with or without timezone (e.g., 2006-01-02T15:04:05Z07:00, 2006-01-02 15:04:05), 12-hour AM/PM with or without seconds (e.g., 03:04:05 PM, 03:04PM), and 24-hour formats (e.g., 15:04:05, 15:04)`

Step 2. Okay, the user asked, "if you're nine years old and in 2030 how old will you be?" Let me think. The current time is October 5, 2025, according to the tool response. So, if today is 2025, then in 2030, the user would be 9 + 5 = 14 years old. Wait, but the user said "in 2030," so from 2025 to 2030 is 5 years. So 9 + 5 = 14. But the user might have meant their own age, but they said "you" which could be a typo. However, the tools don't have a way to calculate age unless the current year is known. The get_current_context tool was used to find the current time, which is 2025. So the answer is 14. But the user might not know the current year, so I need to explain that the current year is 2025 and the answer is 14. But the tools can't provide the current year unless the user's location is known. Since the tool response shows the current time as 2025, I can use that to calculate the age. So the final answer is 14.
Tool Call: `{"name": "response_gen", "arguments": {"response": "If you're nine years old in 2025, you will be 14 years old in 2030."}}`

Figure 12: Step-by-step reasoning and tool usage under PORTool with Qwen-3-1.7B backbone.