TRACE: A Unified Rollout Budget Allocation Framework for Efficient Agentic Reinforcement Learning

Anonymous ACL submission

Abstract

Reinforcement learning with verifiable rewards (RLVR) is a promising approach for enhancing reasoning and agentic behavior in large language models. However, rollout-intensive policy optimization is often limited by insufficient reward contrast, arising from overly simple or complex prompts to generate low-variance feedback, alongside outcome-only rewards that assign the same terminal assessment to every decision in a multi-turn rollout. Past efforts have focused on allocating additional rollout resources to promising prompts, yet they typically operate at the prompt level and do not exploit the prefix structure exposed within multi-turn rollouts. This work targets multi-turn agentic RL by defining each thought-action-observation turn as a distinct node, allowing budget allocation to extend from prompt roots to turn-level prefixes with further continuations. We introduce Tree Rollout Allocation for Contrastive Exploration (TRACE), a unified rollout allocation framework that enhances reward comparisons within a fixed sampling budget. Technically, TRACE prioritizes both prompt roots and intermediate prefixes, focusing on anchors likely to produce diverse outcomes. The resulting adaptive tree structure enriches outcome-only feedback and amplifies the policy-update signal. Empirically, TRACE achieves competitive performance and efficiency gains on typical agentic benchmarks, e.g., improving Qwen3-14B Multi-Hop QA average accuracy by 2.8 points over competitive baselines at equal sampling costs.

1 Introduction

Reinforcement learning with verifiable rewards (RLVR) has become a key mechanism for incentivizing reasoning and agentic behavior in large language models (LLMs), using verifiable feedback to shape exploration over solution paths and environment interactions (Jaech et al., 2024; Guo et al., 2025a). Despite its success, RLVR is computationally expensive because rollouts require long Chain-of-Thought (CoT) (Wei et al., 2022) generation for reasoning and interleaved environment interactions for agentic tasks. Deciding where to spend limited rollout budget is therefore a central design problem (Zheng et al., 2025; Lin et al., 2025).

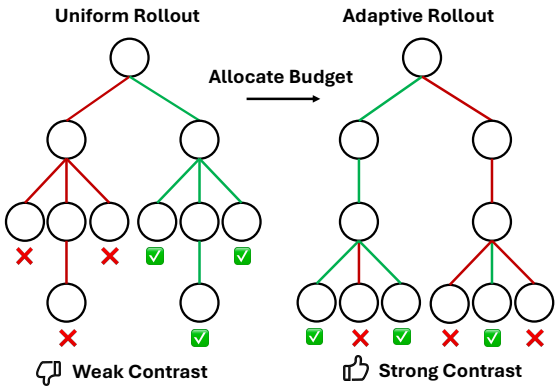


Figure 1: **The intuition behind TRACE.** Same rollouts, more mixed-reward contrast.

In practice, most scalable RLVR pipelines rely on outcome rewards rather than process rewards, avoiding hand-crafted intermediate reward rules that are vulnerable to reward hacking (Shao et al., 2024). The learning signal provided by such rollouts, however, is highly uneven. At the prompt level, overly easy or hard instances tend to produce low-variance outcome rewards (Yu et al., 2025). At the trajectory level, assigning a single terminal reward to a long rollout leaves little local contrast for credit assignment across generated tokens and agent turns (Lightman et al., 2024). Prior work alleviates prompt-level inefficiency through prompt selection (Zheng et al., 2025; Gao et al., 2025; Bae et al., 2026), which forecasts informative prompts to guide rollout allocation (Li et al., 2025c), i.e., determining the sample number to each prompt. Together, they can be unified as scoring a candidate set and assigning rollout budget toward prompts likely to produce contrastive outcome variation: a

prompt assigned zero rollouts is skipped, whereas a selected prompt assigned a larger positive budget receives more independent root rollouts. Yet this allocation strategy acts only at the root. Once a prompt receives budget, each rollout is still generated as an atomic trajectory.

Multi-turn agentic RL exposes a finer allocation unit, the turn. While single-turn reasoning can be segmented at token or sentence boundaries, ReAct-style (Yao et al., 2022) interaction naturally packages each decision as a thought-action-observation node. A complete rollout therefore exposes semantically meaningful prefixes that can be revisited for branching. These visited prefixes become candidate branching points for extra exploration. Some prefixes are unlikely to produce new outcome variation, whereas others are likely to yield counterfactual continuations with different final rewards. Technically, allocating budget at this level is not a matter of sampling more independent trajectories; it is a choice of which prefix occurrences should receive additional continuation branches.

The above viewpoint naturally turns rollout collection into a tree generation. Prompt roots are depth-zero anchors and internal prefixes are non-root turn-level anchors, so root-level allocation becomes a special case of allocating budget over tree nodes. The shared principle is *mixed-reward contrast construction*: allocate rollout budget to anchors whose descendant sets are most likely to contain both successful and failed outcomes. At the root, this principle unifies prompt filtering and rollout-count allocation, with $m_i = 0$ rejecting a prompt and $m_i \geq 2$ setting its root rollout count. Inside an active prompt, the same principle allocates extra branches to prefixes likely to produce opposite-outcome siblings.

We instantiate this idea as **T**ree **R**ollout **A**llocation for **C**ontrastive **E**xploration (**TRACE**), a unified budget-allocation framework for efficient RLVR in multi-turn agentic tasks. TRACE first performs global root allocation over a large candidate pool, then performs prompt-local tree expansion over the prefix occurrences exposed by the resulting rollouts. The resulting trees surface prefix-level contrasts from scalable outcome rewards, providing node-level or stepwise training signals for tree-aware policy optimizers. Across mathematical reasoning, multi-hop QA, and function calling, TRACE improves accuracy over random sampling under the same rollout budget by directing samples toward anchors with useful reward contrast.

In primary, this work's contribution is three-fold:

1. We formulate *mixed-reward contrast allocation* as a unified view of sample-efficient RLVR, incorporating prompt filtering, rollout-count allocation, and prefix-level branching as strategic budget decisions over rollout-tree anchors.
2. We derive *root and prefix allocation utilities* based on a shared descendant-set objective, distributing budget according to the likelihood that an anchor's descendants will yield both successful and failed outcomes.
3. We instantiate this view in **TRACE**, a budget-constrained tree rollout framework that integrates global root allocation with local prefix expansion, enhancing implicit credit signals for sample efficient policy optimization.

2 Preliminaries

2.1 Multi-Turn RLVR as Prefix Histories

This work investigates multi-turn agentic reinforcement learning within the ReAct framework (Yao et al., 2022). We define the input prompt as the root node x of the rollout tree. At each turn $t = 1, \ldots, T$, the model produces a thought τ_t and an action a_t, to which the environment responds with an output o_t. We then encapsulate the complete turn as $n_t := \langle \tau_t, a_t, o_t \rangle$. The history available after turn t is described as

$$\mathcal{H}_t := (x, n_1, \ldots, n_t), \qquad \mathcal{H}_0 := x. \tag{1}$$

The dynamics of the rollout can be expressed in the following hierarchical generative process:

$$(\tau_t, a_t) \sim \pi_\theta(\cdot \mid \mathcal{H}_{t-1}), \tag{2a}$$

$$o_t \sim P_{\text{env}}(\cdot \mid \mathcal{H}_{t-1}, \tau_t, a_t). \tag{2b}$$

A complete rollout is identified by its terminal history $\mathcal{H}_T$, and the RLVR framework assigns a binary terminal reward $r(\mathcal{H}_T) \in \{0, 1\}$. Thus, the underlying training objective is formulated as

$$\max_\theta \; \mathbb{E}_{x \sim \mathcal{D},\, \mathcal{H}_T \sim \mathbb{P}_{\pi_\theta}(\cdot|x)} \big[r(\mathcal{H}_T) \big]. \tag{3}$$

For any prefix $\mathcal{H}_t$, we define the conditional success probability of the prompt problem-solving as

$$V_t^\pi := \mathbb{E}_\pi \big[r(\mathcal{H}_T) \mid \mathcal{H}_t \big]. \tag{4}$$

A prompt is the shortest history $\mathcal{H}_0$, while later prefixes add the thoughts, actions, and observations already produced during the rollout.

2.2 Initialize-then-Expand Rollout Trees

Tree rollout methods commonly follow an initialize-then-expand construction (Ji et al., 2025; Hou et al., 2025). After an allocation rule assigns active prompt x_i a root rollout count m_i, Stage 1 samples complete bare rollouts. For rollout j, let $T_{i,j}$ be its terminal turn index and let $\mathcal{H}_{i,j,t}$ denote its prefix after turn t:

$$\mathcal{H}_{i,j,T_{i,j}} \sim \pi_\theta(\cdot \mid x_i),\ j = 1, \ldots, m_i, \quad (5)$$

and records terminal rewards $r_{i,j}$. Each visited prefix $\mathcal{H}_{i,j,t}$ can then serve as a branching anchor: if prefix expansion assigns $K_{i,j,t}$ continuations to it, Stage 2 samples terminal suffixes

$$S^{(b)}_{i,j,t} \sim \pi_\theta(\cdot \mid \mathcal{H}_{i,j,t}),\ b = 1, \ldots, K_{i,j,t}. \quad (6)$$

The original suffix is the factual branch under the prefix, while newly sampled suffixes $S^{(b)}_{i,j,t}$ are counterfactual branches. This abstraction makes prompt roots and internal prefixes comparable budget-allocation anchors: both are conditioning contexts with terminal descendants below them. For an active prompt x_i, the corresponding nonterminal prefix-anchor index set is

$$\mathcal{A}_i := \{(j,t) : 1 \le j \le m_i,\ 1 \le t < T_{i,j}\}, \quad (7)$$

which excludes terminal leaves and records prefix occurrences available for continuation branching.

3 Rollout Allocation through the Lens of Contrast Construction

In outcome-reward RLVR (Guo et al., 2025a), more rollouts are not automatically more informative. A broad line of process-supervision, preference-learning, and tree-based RL methods reflects the same lesson: learning becomes denser when the model can compare alternatives rather than consume isolated terminal labels (Lightman et al., 2024; Lai et al., 2024; Zhang et al., 2024). If all descendants of an anchor receive the same terminal reward, group-relative or tree-aware updates have little local preference signal (Shao et al., 2024; Yang et al., 2025b): sampled continuations do not tell the optimizer which decisions should be preferred under the same conditioning context. Mixed-reward descendants instead hold the anchor fixed while varying the continuation and outcome, turning a terminal scalar reward into an anchor-local comparison. We therefore treat contrast construction as the allocation objective: a rollout budget is useful to the extent that it creates success/failure comparisons below prompt roots or visited prefixes.

3.1 Prefix Information for Better Difficulty Prediction

Prior sample-efficient RLVR methods use prompt-level difficulty or correctness prediction to filter prompts and allocate root rollouts (Zheng et al., 2025; Gao et al., 2025; Bae et al., 2026; Li et al., 2025c). This motivates the prediction problem in our tree setting: if prompts can be scored before sampling, then visited prefixes should also be scored before assigning continuation budget. Prefixes contain at least the information available at the prompt, and often more, so they provide a richer context for predicting group correctness and downstream contrast.

Proposition 1 (Prefix information improves group difficulty prediction)**.** *For $m \ge 1$, let $\bar{R}_{t,m}$ be the average terminal reward of m independent continuations sampled from prefix $\mathcal{H}_t$. For any squared-loss predictor f that maps a history to a scalar estimate, define $\mathcal{E}^\star_{t,m} := \inf_f \mathbb{E}[(\bar{R}_{t,m} - f(\mathcal{H}_t))^2]$. Then $\mathbb{E}[\bar{R}_{t,m} \mid \mathcal{H}_t] = V^\pi_t$, and for any $t < T$,*

$$\mathcal{E}^\star_{t+1,m} \le \mathcal{E}^\star_{t,m}. \quad (8)$$

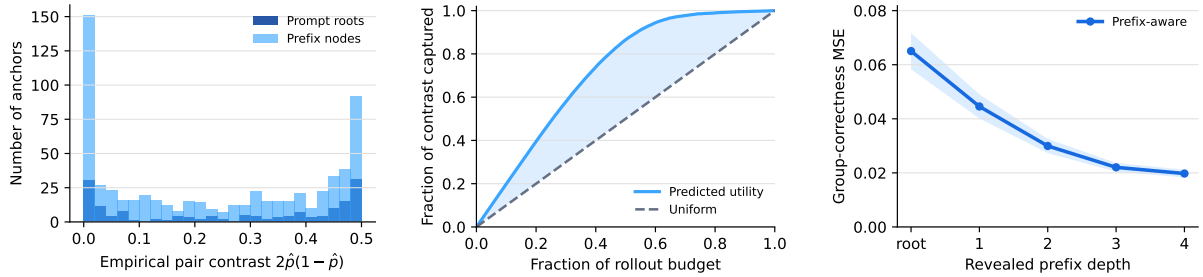
Consequently, $\mathcal{E}^\star_{t,m} \le \mathcal{E}^\star_{0,m}$ for all t. The cases $t = 0$ and $m = 1$ recover prompt-level difficulty prediction and single-rollout reward prediction, respectively.

Proposition 1 formalizes why prediction enters the allocation problem: features derived solely from prompts can support difficulty prediction, while prefix histories offer additional information for predicting group correctness and distributing continuation budgets. We attach the corresponding proof in Appendix D.1.

3.2 A Shared Mixed-Outcome Principle for Roots and Prefixes

Given such root- and prefix-level predictions, the allocation target is the same at both scales: at a prompt root, the allocator asks whether fresh rollouts will form a diverse outcome group; at a visited prefix, it asks whether additional continuations from the same history can disagree with the factual suffix. This gives a single contrast-seeking rule: anchors near deterministic success or failure provide little contrast, while anchors with intermediate conditional success probability are more likely to expose both successful and failed descendants.

This prefix-level view explains how outcome-only rewards can still induce local credit. Shared-prefix branches keep the past fixed and vary only



(a) Contrast heterogeneity (b) Allocation efficiency (c) Prefix prediction

Figure 2: **Contrastive-allocation diagnostics.** Prompt roots and prefix nodes exhibit heterogeneous pair-contrast scores, so ordering anchors by predicted contrast utility captures useful reward disagreement much faster than uniform allocation. Prefix information also improves group-correctness prediction as interaction history is revealed.

the future continuation, so opposite terminal outcomes become a local comparison for tree-aware optimizers. From a martingale view of conditional success, this remaining contrast has a simple form.

Proposition 2 (Prefix Uncertainty as Remaining Contrast Potential)**.** *Let $\mathcal{F}_t := \sigma(\mathcal{H}_t)$, $Z_t := V_t^\pi = \mathbb{E}_\pi[r(\mathcal{H}_T) \mid \mathcal{F}_t]$, and $[Z]_{t:T} := \sum_{s=t}^{T-1}(Z_{s+1} - Z_s)^2$. For any prefix $\mathcal{H}_t$,*

$$\mathbb{E}_\pi\left[[Z]_{t:T} \mid \mathcal{F}_t\right] = V_t^\pi\left(1 - V_t^\pi\right). \tag{9}$$

Proposition 2 shows that $V_t^\pi(1-V_t^\pi)$ is not only Bernoulli uncertainty but the expected remaining variation in success probability below the prefix. It therefore measures downstream contrast potential. Fig. 2 confirms the corresponding empirical picture: prefix information improves group-correctness prediction, and a small subset of anchors accounts for much of the available reward disagreement.

3.3 Contrast Values as Update Activation

The allocation rules are chosen before the optimizer sees gradients, but they target the activation frontier where gradients become informative. Under binary rewards, a root or prefix anchor activates local gradient contrast only when its descendants expose both success and failure.

This observation connects contrast allocation to optimizer-visible gradients. Let $\mathcal{A}_{\text{root}} = \{x_i\}_{i=1}^B$ and let $\mathcal{A}_{\text{pref}}$ denote the local prefix-anchor set $\mathcal{A}_i$ for an active prompt; aggregation over prompts simply sums these local terms. For stage $q \in \{\text{root}, \text{pref}\}$ and allocation b, write the aggregate local policy-gradient contribution as

$$\begin{aligned} G_q(b) &= \sum_{h \in \mathcal{A}_q} \mathbb{I}\{\mathsf{Act}(h, b_h)\}\widetilde{G}_h(b_h), \\ \widetilde{G}_h(b_h) &= \sum_{\mathcal{H} \in \mathcal{D}_h(b_h)} \omega_{\mathcal{H}} \nabla_\theta \log \pi_\theta(\mathcal{H} \mid h), \end{aligned} \tag{10}$$

Here $\mathcal{D}_h(b_h)$ is the sampled terminal-descendant set below anchor h, $\omega_{\mathcal{H}}$ is the optimizer-dependent contrast weight, and Act is the event that descendants below h contain both outcomes.

Proposition 3 (Activation allocation dominates uniform)**.** *Let $b_q^\star$ be the TRACE allocation for stage $q \in \{\text{root}, \text{pref}\}$, and let b_q^u be any feasible uniform allocation at the same budget. Write $G_q^\star := G_q(b_q^\star)$ and $G_q^u := G_q(b_q^u)$. Assume normalized conditional gradient energy, meaning $\mathbb{E}[\|\widetilde{G}_h(b_h)\|^2 \mid \mathsf{Act}(h, b_h), h] = c_q > 0$ for all anchors in stage q. If the combined update also satisfies $\mathbb{E}\langle G_{\text{root}}^\star, G_{\text{pref}}^\star\rangle \geq \mathbb{E}\langle G_{\text{root}}^u, G_{\text{pref}}^u\rangle$, then*

$$\begin{aligned} &\mathbb{E}\|G_q^\star\|^2 \geq \mathbb{E}\|G_q^u\|^2, \quad q \in \{\text{root}, \text{pref}\}, \\ &\mathbb{E}\|G_{\text{root}}^\star + G_{\text{pref}}^\star\|^2 \geq \mathbb{E}\|G_{\text{root}}^u + G_{\text{pref}}^u\|^2. \end{aligned} \tag{11}$$

The first line shows that each stage alone dominates its uniform counterpart; the second line gives the combined gradient-norm dominance. Inequalities are strict when uniform is not optimal for the corresponding stage.

Proposition 3 suggests the main allocation consequence: both stages separately strengthen the expected squared local gradient signal over uniform allocation, and their combination strengthens the full tree update when the two gradient sources do not cancel. The proof in Appendix D.3 shows the factorization behind this statement: under standard outcome-reward updates, the local gradient contribution is zero unless opposite outcomes are observed below the same anchor, so the expected squared gradient norm equals an activation probability times a conditional gradient scale.

4 TRACE

The previous section identifies the useful sampling targets: roots and prefixes whose descendants are

Algorithm 1: TRACE rollout allocation

Input: Candidate pool sequence $\{\mathcal{B}_s\}_{s=1}^{S}$; initial policy π_θ; predictor $\tilde{V}_\psi$; root budget M; expansion factor N.
Output: Finetuned LLM policy π_θ.
for $s = 1$ **to** S **do**
  Initialize collected tree set $\mathcal{G}_s \leftarrow \emptyset$;
  `// Global root allocation`
  Estimate $\tilde{V}_\psi(x_i)$ for all $x_i \in \mathcal{B}_s$;
  Solve Eq. (13) for root counts $\{m_i\}_{i=1}^{B}$;
  foreach *prompt* x_i *with* $m_i > 0$ **do**
    Generate m_i bare rollouts from x_i and collect terminal rewards;
    Build the prefix-anchor index set $\mathcal{A}_i$;
    Estimate $\tilde{V}_\psi(\mathcal{H}_{i,j,t})$ for each $(j,t) \in \mathcal{A}_i$;
    `// Local prefix expansion`
    Solve Eq. (17) for continuation counts $\{K_{i,j,t}\}$;
    foreach (j,t) *with* $K_{i,j,t} > 0$ **do**
      Generate $K_{i,j,t}$ continuations from prefix $\mathcal{H}_{i,j,t}$;
    Add the resulting prompt-local rollout trees to $\mathcal{G}_s$;
  `// Predictor and policy updates`
  Compute recursive targets $\widehat{V}$ on $\mathcal{G}_s$ and update ψ by Eq. (20);
  Update π_θ with any tree-aware optimizer using $\mathcal{G}_s$;

likely to disagree in terminal reward. TRACE instantiates this principle as a global-to-local allocation procedure, summarized in Fig. 3 and Algorithm 1. It solves root allocation over a candidate prompt pool and local prefix expansion over visited prefix occurrences. The predictor $\tilde{V}_\psi(\mathcal{H})$ estimates the same conditional success probability written as V_t^π for a turn-indexed history $\mathcal{H}_t$ in Section 2. The resulting trees supervise this predictor through recursive targets and provide root-/prefix-level comparisons for tree-aware policy optimization.

4.1 Global Root Allocation

At step s, TRACE allocates a root rollout budget over a candidate prompt set $\mathcal{B}_s = \{x_1, \ldots, x_B\}$, where each prompt receives a count in $\mathcal{M} := \{0\} \cup \{2, \ldots, M\}$. Let $v_i := \tilde{V}_\psi(x_i)$ be the predicted root success probability. For $m \geq 2$, TRACE assigns

$$V_{\text{root}}(x_i, m) = 1 - v_i^m - (1 - v_i)^m, \tag{12}$$

and sets $V_{\text{root}}(x_i, 0) = V_{\text{root}}(x_i, 1) = 0$. This is the predicted probability that m root rollouts for x_i contain success and failure. In math, root allocation solves the budget-constrained problem:

$$\begin{aligned} &\max_{m_1,\ldots,m_B} \sum_{i=1}^{B} V_{\text{root}}(x_i, m_i) \\ &\text{s.t.,} \sum_{i=1}^{B} m_i = M,\ m_i \in \mathcal{M}. \end{aligned} \tag{13}$$

The zero count skips a prompt, while any $m_i \geq 2$ selects it and determines how many bare rollouts are generated.

4.2 Local Prefix Expansion

After active prompt x_i finishes its m_i bare rollouts, let $r_{i,j} \in \{0, 1\}$ be the terminal reward of bare rollout j. Its prefix after turn t is $\mathcal{H}_{i,j,t}$, with $\mathcal{H}_{i,j,0} = x_i$. The prefix-anchor index set is

$$\mathcal{A}_i := \{(j,t) : 1 \leq j \leq m_i,\ 1 \leq t < T_{i,j}\}. \tag{14}$$

Each index $(j,t) \in \mathcal{A}_i$ denotes a visited nonterminal prefix together with the factual terminal outcome already observed below it. We allocate a fixed local budget

$$\sum_{(j,t)\in\mathcal{A}_i} K_{i,j,t} = m_i N, \tag{15}$$

so Stage 2 does not wait for or compete with other prompts once prompt i is ready.

Using the same predictor for the prefix success probability, assigning k continuations to prefix occurrence (j,t) has value

$$\begin{aligned} V_{\text{pref}}(i,j,t,k) := 1 - \Big[& r_{i,j}\tilde{V}_\psi(\mathcal{H}_{i,j,t}) \\ & + (1 - r_{i,j})(1 - \tilde{V}_\psi(\mathcal{H}_{i,j,t})) \Big]^k, \end{aligned} \tag{16}$$

namely the probability that at least one new continuation flips the observed reward. Local prefix expansion solves

$$\begin{aligned} &\max_{\{K_{i,j,t}\}_{(j,t)\in\mathcal{A}_i}} \sum_{(j,t)\in\mathcal{A}_i} V_{\text{pref}}(i,j,t,K_{i,j,t}) \\ &\text{s.t.} \sum_{(j,t)\in\mathcal{A}_i} K_{i,j,t} = m_i N. \end{aligned} \tag{17}$$

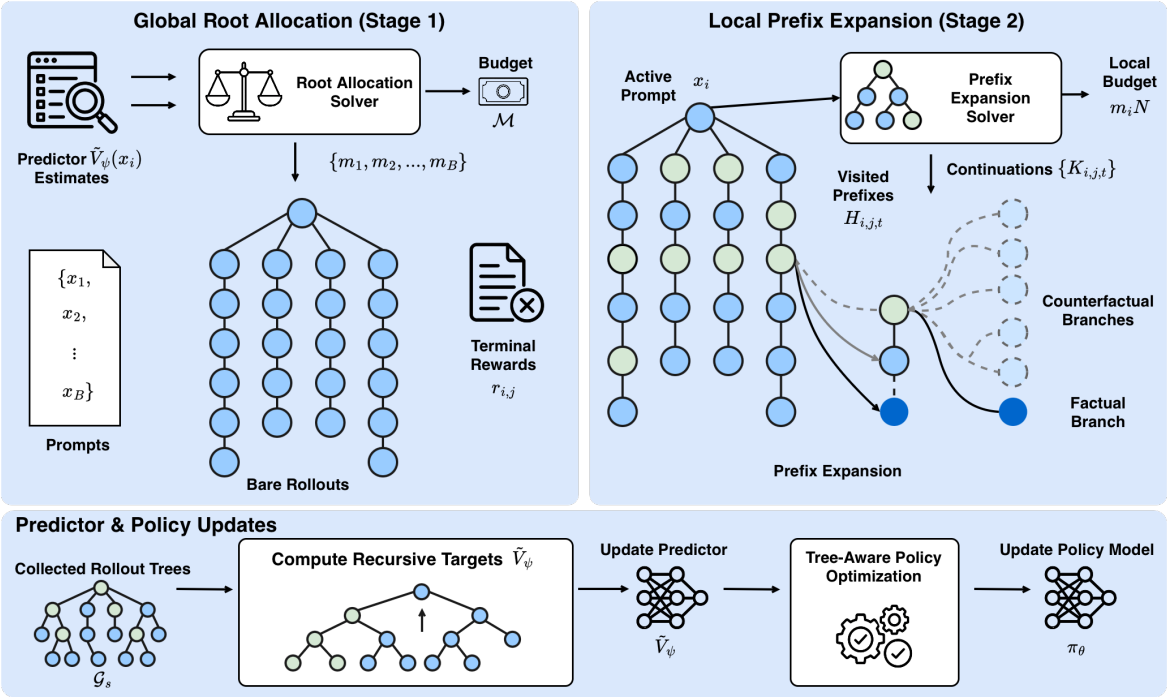


Figure 3: **Framework overview of TRACE.** A prefix value predictor scores prompt roots and visited prefixes, TRACE solves budgeted root allocation and prompt-local prefix expansion, and the resulting rollout trees provide recursive value targets and root-/prefix-level comparisons for tree-aware policy optimization.

Both allocation problems are solved by dynamic programming, whose cost is negligible relative to rollout generation. Expanding (j, t) fixes the prefix $\mathcal{H}_{i,j,t}$ and resamples a new continuation after that prefix. Terminal leaves are not expanded.

This factorization is intentionally system-aware: a fully global prefix allocator over $\bigcup_i \mathcal{A}_i$ would wait for all active prompts to finish bare rollouts before launching any continuation. TRACE only requires prompt-level completion, so expansion for prompt i can be enqueued once its own m_i bare rollouts have returned, without assuming that those rollouts were co-located on the same worker, thereby largely mitigating long-tail waiting.

4.3 Online Prefix Value Estimation

The allocation objectives require the conditional success probability at both prompt roots and visited prefixes. We learn a lightweight predictor

$$\tilde{V}_\psi : \mathcal{H}_t \mapsto \tilde{V}_\psi(\mathcal{H}_t) \in [0, 1], \tag{18}$$

which provides $\tilde{V}_\psi(x_i)$ for Eq. (12) and $\tilde{V}_\psi(\mathcal{H}_{i,j,t})$ for Eq. (16). It is used only by the allocator, not by the downstream policy optimizer.

Recursive tree-backed targets. After a rollout tree is collected, its terminal leaves provide binary rewards. For any node y, let $\mathcal{C}(y)$ be its children and let n_y be the number of executed terminal descendants below it:

$$n_y = \begin{cases} 1, & y \text{ is a terminal leaf}, \\ \sum_{c \in \mathcal{C}(y)} n_c, & \text{otherwise}, \end{cases}$$

We set $\widehat{V}(y) = r_y$ at terminal leaves and compute internal targets by a bottom-up empirical average:

$$\widehat{V}(y) = \frac{1}{n_y} \sum_{c \in \mathcal{C}(y)} n_c \widehat{V}(c). \tag{19}$$

Thus $\widehat{V}(y)$ is the empirical success rate of terminal descendants below y. We train $\tilde{V}_\psi$ by mean-squared regression over a supervision set $\mathcal{S}$ of roots and informative internal nodes:

$$\mathcal{L}_{\text{value}} := \frac{1}{|\mathcal{S}|} \sum_{y \in \mathcal{S}} \left(\tilde{V}_\psi(y) - \widehat{V}(y) \right)^2. \tag{20}$$

The exact supervision set and predictor implementation details are reported in Appendix E.6.

4.4 Consuming TRACE Trees

TRACE outputs rollout trees with terminal rewards and visited internal decisions, then passes these trees to a tree-aware policy optimizer as in Algorithm 1. This optimizer can use any prefix-level credit rule that propagates leaf rewards through the

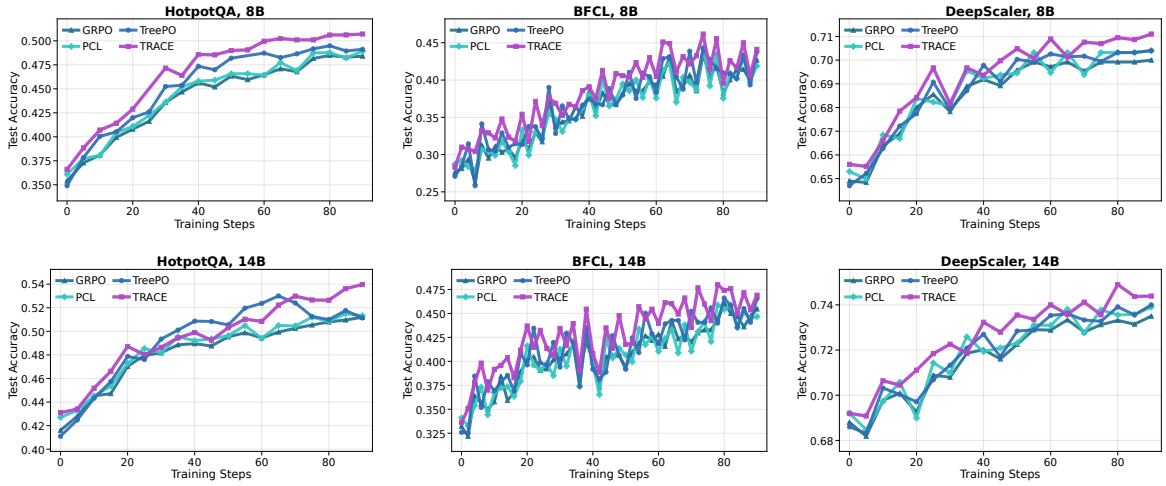


Figure 4: **Test accuracy during training.** We compare GRPO, PCL, random TreePO allocation, and TRACE under the same rollout-budget setting. Higher curves indicate stronger final policies under identical sampling budgets.

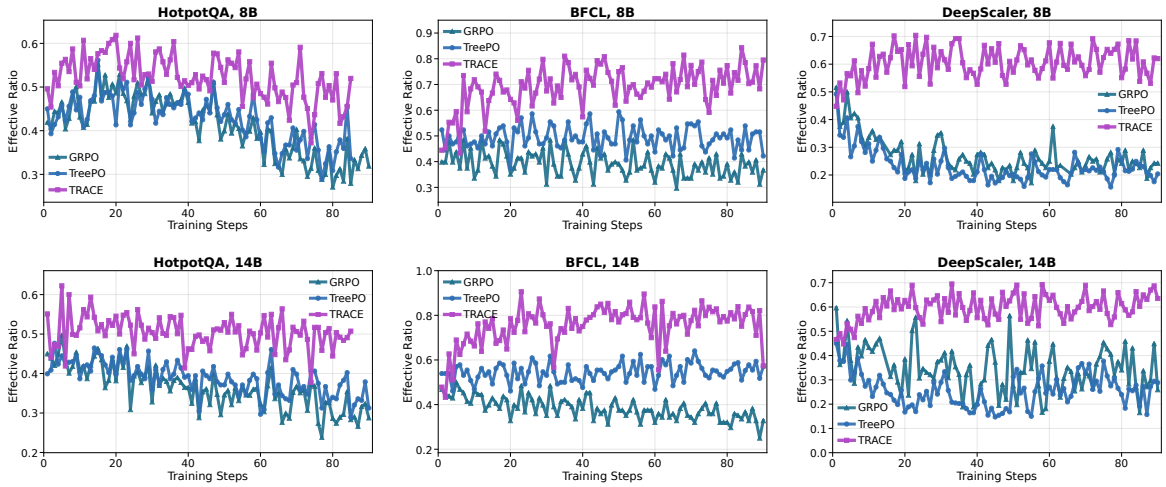


Figure 5: **Effective ratio during training.** The panels compare the fraction of contrastive samples selected by each allocation strategy. Higher values indicate more non-degenerate reward groups per update.

tree, including process-reward backups or group-relative backups. In our experiments, we use the concrete instance described in Appendix E.4.

5 Experiments

5.1 Experimental Setup

Datasets and models. We evaluate TRACE on three multi-turn domains: DeepScaler mathematical reasoning with a Python interpreter (Luo et al., 2025), retrieval-based Multi-Hop QA trained on HotpotQA and tested on HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023), and BFCL v4 multi-turn function calling (Patil et al., 2025). Main experiments use Qwen3-8B and Qwen3-14B backbones (Yang et al., 2025a).

Baselines. We compare against ReAct (Yao et al., 2022), GRPO (Shao et al., 2024), PCL prompt selection (Gao et al., 2025), and TreePO, which uses tree-structured rollouts and tree-aware updates but allocates continuation budget uniformly. TRACE differs by actively allocating budget at both prompt roots and internal prefixes. All training comparisons use the same rollout-budget accounting, so gains reflect allocation rather than additional samples.

Metrics. We report final-answer accuracy for DeepScaler, answer-match reward for Multi-Hop QA, and official success rate for BFCL. Appendix C reports allocation and predictor diagnostics, while Appendix E gives full dataset, model, optimizer, and rollout details.

5.2 Main Results

5.2.1 Improved Accuracy under same Budget

Figure 4 shows that TRACE consistently improves late-stage accuracy across math reasoning, retrieval-based QA, and function calling. The comparison with TreePO is especially revealing: both methods use tree rollouts and tree-aware updates, but TRACE allocates branches to roots and prefixes predicted to be more informative, yielding stronger accuracy under the same rollout budget rather than larger sampling volume. On DeepScaler, it raises the in-domain average over GRPO from 70.0 to 71.1 for Qwen3-8B and from 73.5 to 74.9 for Qwen3-14B, with similar gains on Multi-Hop QA and BFCL. Full benchmark breakdowns appear in Appendix C.1.

5.2.2 Sample Efficiency

Figure 5 shows the allocation mechanism directly: TRACE turns the same rollout budget into more informative samples for each update. By avoiding saturated roots and unproductive prefixes, it creates more mixed-outcome groups for policy updates, making each rollout budget unit carry denser learning signal. Appendix C.1 provides the metric definition and full diagnostic curves.

5.2.3 Reliable Difficulty Prediction

Figures 6 and 7 show that the lightweight predictor learns useful rankings at both prompt and prefix levels. The prefix result is the more diagnostic one: it indicates that partial interaction histories contain learnable local uncertainty, enough to guide branching before the final reward is observed or expansion begins. Appendix C further reports the Spearman definition and per-domain trends.

5.3 Additional Analysis

5.3.1 Ablation Studies

We isolate the two allocation stages on Qwen3-8B HotpotQA by replacing each active allocator with uniform allocation while keeping the tree budget and optimizer fixed. Table 1 shows that both stages help and their gains stack: root allocation selects prompts likely to become non-degenerate, while prefix allocation spends continuation budget on prefixes where additional descendants can still reveal contrast.

5.3.2 Compatibility to Different Budget

We also vary the tree budget on Qwen3-8B HotpotQA. Since the expected rollout budget is $M(1+N/2)$, $(512, 2)$ uses budget 1024, while $(512, 6)$ and $(1024, 2)$ both use budget 2048. Table 2 shows that TRACE improves TreePO at every budget, lifting the effective ratio by about 9.5–10.2 points and gaining roughly one accuracy point. It also shows that budget shape matters, not only total size: at budget 2048, broader root coverage with $(1024, 2)$ is stronger than deeper continuation sampling from 512 roots. The bottleneck is therefore not just rollout count, but whether the budget reaches states where rewards can form contrast.

Table 1: **Ablation of active allocation stages on Qwen3-8B HotpotQA.** Stage 1 allocates prompt-root budget; Stage 2 allocates continuation budget over visited prefixes.

Stage 1	Stage 2	Avg. Acc.	Avg. Eff. Ratio
Uniform	Uniform	49.5	42.8
Active	Uniform	49.8	49.1
Uniform	Active	50.0	47.3
Active	Active	**50.6**	**52.3**

Table 2: **Compatibility to different budgets on Qwen3-8B HotpotQA.** We vary the root budget M and continuation budget N and compare random TreePO allocation with TRACE.

M	N	Method	Avg. Acc.	Avg. Eff. Ratio
512	2	TreePO	48.8	32.2
512	2	TRACE	49.7	42.4
512	6	TreePO	49.4	37.7
512	6	TRACE	50.3	47.8
1024	2	TreePO	49.5	42.8
1024	2	TRACE	**50.6**	**52.3**

6 Conclusion

We presented TRACE, a tree rollout allocation framework for multi-turn agentic RLVR. TRACE turns rollout collection into a budgeted choice over prompt roots and internal prefixes, using the same uncertainty signal to find anchors likely to expose reward contrast. This unifies prompt selection, rollout-count allocation, and prefix expansion under one objective while staying compatible with tree-aware policy optimization. Across math, retrieval, and tool-use agents, TRACE improves training efficiency without changing the verifier or reward format. The broader message is simple: in agentic RL, where to branch is as important as how many rollouts to sample.

Limitations

TRACE assumes automatically verifiable outcomes and reusable prefix histories, so settings with noisy human preferences, non-stationary tools, or very long memory may require adapting the contrast utility. Our current predictor is lightweight, but prefix scoring is not yet fully batched or cached, leaving room for systems optimization beyond the overhead study in Appendix C.5. Finally, broader sweeps over backbone families, tools, and latency regimes would further refine the deployment recipe.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight experience replay. *Advances in neural information processing systems*, 30.

Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. 2019. Rudder: Return decomposition for delayed rewards. *Advances in Neural Information Processing Systems*, 32.

Sanghwan Bae, Jiwoo Hong, Min Young Lee, Hanbyul Kim, JeongYeon Nam, and Donghyun Kwak. 2026. Online difficulty filtering for reasoning oriented reinforcement learning. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 700–719.

Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. 2025. Self-evolving curriculum for llm reasoning. *arXiv preprint arXiv:2505.14970*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, and 1 others. 2025. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, and 1 others. 2025. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*.

Mehdi Fatemi, Banafsheh Rafiee, Mingjie Tang, and Kartik Talamadupula. 2025. Concise reasoning via reinforcement learning. *arXiv preprint arXiv:2504.05185*.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2023. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*.

Zhaolin Gao, Joongwon Kim, Wen Sun, Thorsten Joachims, Sid Wang, Richard Yuanzhe Pang, and Liang Tan. 2025. Prompt curriculum learning for efficient llm post-training. *arXiv preprint arXiv:2510.01135*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025a. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Yiran Guo, Lijie Xu, Jie Liu, Dan Ye, and Shuang Qiu. 2025b. Segment policy optimization: Effective segment-level credit assignment in rl for large language models. *Advances in Neural Information Processing Systems*, 38:114399–114431.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, and 1 others. 2024a. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850.

Mingqian He, Yongliang Shen, Wenqi Zhang, Zeqi Tan, and Weiming Lu. 2024b. Advancing process verification for large language models via tree-based preference learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2086–2099.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625.

Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. 2025. Treerl: Llm reinforcement learning with on-policy tree search. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12355–12369.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. 2025a. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*.

Zengjie Hu, Jiantao Qiu, Tianyi Bai, Haojin Yang, Binhang Yuan, Qi Jing, Conghui He, and Wentao Zhang. 2025b. Vade: Variance-aware dynamic sampling via online sample-level difficulty estimation for multimodal rl. *arXiv preprint arXiv:2511.18902*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

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

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pages 6769–6781.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. 2024. Vineppo: Unlocking rl potential for llm reasoning through refined credit assignment.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. 2024. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*.

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. 2024. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms. *arXiv preprint arXiv:2406.18629*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857.

Pengxiang Li, Zhi Gao, Bofei Zhang, Yapeng Mi, Xiaojian Shawn Ma, Chenrui Shi, Tao Yuan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, and 1 others. 2025a. Iterative tool usage exploration for multimodal agents via step-wise preference tuning. *Advances in Neural Information Processing Systems*, 38:59496–59528.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025b. Limr: Less is more for rl scaling. *arXiv preprint arXiv:2502.11886*.

Ziniu Li, Congliang Chen, Tianyun Yang, Tian Ding, Ruoyu Sun, Ge Zhang, Wenhao Huang, and Zhi-Quan Luo. 2025c. Knapsack rl: Unlocking exploration of llms via optimizing budget allocation. *arXiv preprint arXiv:2509.25849*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.

Zhihang Lin, Mingbao Lin, Yuan Xie, and Rongrong Ji. 2025. Cppo: Accelerating the training of group relative policy optimization-based reasoning models. *Advances in Neural Information Processing Systems*, 38:61043–61068.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. 2025. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint arXiv:2505.24864*.

Jieyi Long. 2023. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Li Erran Li, and 1 others. 2025. Deepscaler: Surpassing o1-preview with a 1.5 b model by scaling rl. *Notion Blog*, 3(5).

Yixiu Mao, Yun Qu, Qi Wang, Heming Zou, and Xiangyang Ji. 2026. Dynamics-predictive sampling for active rl finetuning of large reasoning models. *arXiv preprint arXiv:2603.10887*.

Fanqing Meng, Lingxiao Du, Zongkai Liu, Zhixiang Zhou, Quanfeng Lu, Daocheng Fu, Botian Shi, Wenhai Wang, Junjun He, Kaipeng Zhang, and 1 others. 2025. Mm-eureka: Exploring visual aha moment with rule-based large-scale reinforcement learning. *CoRR*.

Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. 2025. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language

models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711.

Yun Qu, Qi Wang, Yixiu Mao, Vincent Tao Hu, Björn Ommer, and Xiangyang Ji. 2025. Can prompt difficulty be online predicted for accelerating rl finetuning of reasoning models? *Preprint*, arXiv:2507.04632.

Yun Qu, Qi Wang, Yixiu Mao, Heming Zou, Yuhang Jiang, Weijie Liu, Clive Bai, Kai Yang, Yangkun Chen, Saiyong Yang, and 1 others. 2026. Small generalizable prompt predictive models can steer efficient rl post-training of large reasoning models. *arXiv preprint arXiv:2602.01970*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*.

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

Philip Sedgwick. 2014. Spearman's rank correlation coefficient. *Bmj*, 349.

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

Qianli Shen, Daoyuan Chen, Yilun Huang, Zhenqing Ling, Yaliang Li, Bolin Ding, and Jingren Zhou. 2025. Bots: A unified framework for bayesian online task selection in llm reinforcement finetuning. *arXiv preprint arXiv:2510.26374*.

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*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Sijun Tan, Michael Luo, Colin Cai, Tarun Venkat, Kyle Montgomery, Aaron Hao, Tianhao Wu, Arnav Balyan, Manan Roongta, Chenguang Wang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. 2025. rllm: A framework for post-training language agents. Notion Blog.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*.

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, and 1 others. 2025. Reinforcement learning for reasoning in large language models with one training example. *arXiv preprint arXiv:2504.20571*.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, and 1 others. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Liang Wen, Yunke Cai, Fenrui Xiao, Xin He, Qi An, Zhenyu Duan, Yimin Du, Junchen Liu, Lifu Tang, Xiaowei Lv, and 1 others. 2025. Light-r1: Curriculum sft, dpo and rl for long cot from scratch and beyond. *arXiv preprint arXiv:2503.10460*.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. 2024. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, and 1 others. 2025. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *International Conference on Learning Representations*, volume 2025, pages 72274–72303.

Haobo Xu, Sirui Chen, Ruizhong Qiu, Yuchen Yan, Chen Luo, Monica Cheng, Jingrui He, and Hanghang Tong. 2026. Prune as you generate: Online rollout pruning for faster and better rlvr. *arXiv preprint arXiv:2603.24840*.

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

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, and 1 others. 2024. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Xiaodan Liang, Yiwei Wang, and Jing Tang. 2025b. Treerpo: Tree relative policy optimization. *arXiv preprint arXiv:2506.05183*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. 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. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Zhiyuan Yao, Yi-Kai Zhang, Yuxin Chen, Yueqing Sun, Zishan Xu, Yu Yang, Tianhao Hu, Qi Gu, Hui Su, and Xunliang Cai. 2026. Coba-rl: Capability-oriented budget allocation for reinforcement learning in llms. *arXiv preprint arXiv:2602.03048*.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244.

Yongcheng Zeng, Zexu Sun, Bokai Ji, Erxue Min, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Haifeng Zhang, Xu Chen, and Jun Wang. 2025. Cures: From gradient analysis to efficient curriculum learning for reasoning llms. *Preprint*, arXiv:2510.01037.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772.

Wenlin Zhang, Xiangyang Li, Kuicai Dong, Yichao Wang, Pengyue Jia, Xiaopeng Li, Yingyi Zhang, Derong Xu, Zhaocheng Du, Huifeng Guo, and 1 others. 2025a. Process vs. outcome reward: Which is better for agentic rag reinforcement learning. *Advances in Neural Information Processing Systems*, 38:58701–58729.

Xiaojiang Zhang, Jinghui Wang, Zifei Cheng, Wenhao Zhuang, Zheng Lin, Minglei Zhang, Shaojie Wang, Yinghan Cui, Chao Wang, Junyi Peng, and 1 others. 2025b. Srpo: A cross-domain implementation of large-scale reinforcement learning on llm. *arXiv preprint arXiv:2504.14286*.

Yi-Kai Zhang, Yueqing Sun, Hongyan Hao, Qi Gu, Xunliang Cai, De-Chuan Zhan, and Han-Jia Ye. 2026a. $v_\{0.5\}$: Generalist value model as a prior for sparse rl rollouts. *arXiv preprint arXiv:2603.10848*.

Zhi Zhang, Zhen Han, Costas Mavromatis, Qi Zhu, Yunyi Zhang, Sheng Guan, Dingmin Wang, Xiong Zhou, Shuai Wang, Soji Adeshina, and 1 others. 2026b. Train less, learn more: Adaptive efficient rollout optimization for group-based reinforcement learning. *arXiv preprint arXiv:2602.14338*.

Yubao Zhao, Weiquan Huang, Sudong Wang, Ruochen Zhao, Chen Chen, Yao Shu, and Chengwei Qin. 2026. Training multi-turn search agent via contrastive dynamic branch sampling. *arXiv preprint arXiv:2602.03719*.

Haizhong Zheng, Yang Zhou, Brian Bartoldson, Bhavya Kailkhura, Fan Lai, Jiawei Zhao, and Beidi Chen. 2025. Act only when it pays: Efficient reinforcement learning for llm reasoning via selective rollouts. *Advances in Neural Information Processing Systems*, 38:124321–124346.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

Appendix Overview

This appendix provides supplementary material for TRACE, including related work, discussion, extended experimental results, theoretical proof, experimental details, and representative data examples. The appendix is organized as follows:

- **Appendix A (Related Work):** reviews prior work on RLVR and agentic RLVR, sample-efficient prompt selection and allocation, tree search for LLMs, and credit assignment under sparse rewards.
- **Appendix B (Discussion):** discusses allocation as an orthogonal layer, the axes of training-time sampling in Table 3, and root-to-prefix predictor generalization.
- **Appendix C (Extended Experimental Results):** reports the full benchmark tables and diagnostic analyses, including effective-ratio behavior, prefix-predictor rank correlation, transfer to Llama-3.2-3B-Instruct, computational overhead, and TRACE allocation behavior.
- **Appendix D (Theoretical Proof):** gives detailed proofs for prefix predictability, the energy identity connecting prefix uncertainty to reward variance, and the TRACE dominance result.
- **Appendix E (Experimental Details):** describes the tasks, models, training settings, policy optimizers and TreePO instances, rollout construction and budget accounting, prefix-predictor supervision and implementation, code availability, and artifact intended use.
- **Appendix F (Data Examples):** presents representative real data examples from the three training domains, including the system prompts and tool schemas seen by the agents.

A Related Work

RLVR and Agentic RLVR. Reinforcement learning with verifiable rewards (RLVR) has become a dominant post-training paradigm for LLMs on tasks with automatically checkable outcomes (Jaech et al., 2024; Guo et al., 2025a; Team et al., 2025). Proximal Policy Optimization (PPO) (Schulman et al., 2017) remains a general policy-gradient backbone, while Group Relative Policy Optimization (GRPO) (Shao et al., 2024) is especially attractive for RLVR because it removes the expensive value model and estimates advantages from groups of sampled rollouts. Much of RLVR still lives in a flat regime where a prompt is sampled, complete responses are generated, and a terminal reward drives the update. Agentic environments fracture this abstraction. A rollout is a chain of thought-action-observation turns (Yao et al., 2022), and intermediate prefixes such as search queries, tool calls, and partial solution states can alter the entire downstream trajectory. TRACE uses this structure to turn rollout budget from a scalar sampling count into a structural allocation decision across roots and prefixes.

Sample Efficient RLVR. Sample efficient RLVR improves policy updates by deciding which examples deserve rollout budget. Offline curation filters prompts by difficulty, diversity, length, or estimated solution quality (Ye et al., 2025; Li et al., 2025b; Wen et al., 2025; Hu et al., 2025a; Yang et al., 2024; Fatemi et al., 2025; Wang et al., 2025), reducing training cost but adding preprocessing and remaining static as the policy evolves (Qu et al., 2025; Gao et al., 2025). Online prompt selection addresses this limitation by adapting to the current model, either through rollout-based filtering of uninformative prompts (Yu et al., 2025; Liu et al., 2025; Cui et al., 2025; Meng et al., 2025; Bae et al., 2026; Xu et al., 2026; Zhang et al., 2026b) or through predictors that estimate prompt difficulty (Zhang et al., 2025b; Zheng et al., 2025; Gao et al., 2025; Qu et al., 2025; Zeng et al., 2025; Chen et al., 2025; Mao et al., 2026; Qu et al., 2026; Zhang et al., 2026a; Hu et al., 2025b; Shen et al., 2025). A complementary prompt-level allocation line asks how many samples each selected prompt should receive (Li et al., 2025c; Yao et al., 2026). This line is effective for dynamic rollout accounting, but it typically relies on observed per-prompt outcomes and has limited ability to generalize to unseen prompts without additional rollouts. TRACE treats filtering, rollout counts and prefix branching as boundary cases of one allocation operation. A lightweight generalizable predictor scores candidate anchors from prompt roots to internal prefixes, and the allocator assigns budget according to predicted reward contrast.

Tree Search for LLMs. Tree search gives LLMs a way to expose latent alternatives rather than commit to a single trajectory. Test-time methods such as Tree-of-Thought (Yao et al., 2023; Long,

Table 3: **Qualitative comparison of training-time allocation methods.** GRPO and PCL operate over flat rollouts, TreePO introduces a tree substrate with random branching, and TRACE uses learned allocation at both prompt roots and visited prefixes.

Method	Prompt Selection	Root Budget	Prefix Expansion	Tree-Structured Update	Learned Allocator
GRPO	Random	Uniform	×	×	×
PCL	Predictive	Fixed	×	×	Root only
TreePO	Random	Uniform	Random	✓	×
Ours	Predictive	Adaptive	Predictive	✓	Root + prefix

2023; Snell et al., 2024; Koh et al., 2024; Zhou et al., 2023) and MCTS-style decoding (Xin et al., 2025) spend additional computation to search for a stronger answer on the current instance. Offline variants sample or score reasoning trees and distill the induced comparisons into SFT or preference-learning data (Feng et al., 2023; He et al., 2024b; Xie et al., 2024; Zhang et al., 2025a; Lai et al., 2024; Li et al., 2025a). Online tree-based RL brings branching into training, using tree rollouts for value estimation or finer credit assignment (Zhang et al., 2024; Hou et al., 2025; Ji et al., 2025; Yang et al., 2025b; Kazemnejad et al., 2024; Guo et al., 2025b; Dong et al., 2025; Zhao et al., 2026). TRACE is closest to this online lineage, but repurposes the tree. It branches over agent interaction prefixes rather than tokens or generic reasoning steps, and uses the tree not as an inference solver, but as an allocation substrate for discovering reward contrast under a fixed rollout budget.

Credit Assignment under Sparse Rewards. Sparse outcome rewards are coarse signals for multi-turn agents because one terminal success or failure will be assigned back to many earlier decisions while intermediate decisions can carry sharply different responsibility for the final result (Feng et al., 2025). Classical RL attacks this with exploration bonuses, replay, goal relabeling, reward shaping, or value-based credit estimates (Pathak et al., 2017; Andrychowicz et al., 2017; Ng et al., 1999; Arjona-Medina et al., 2019). In LLM agents, process reward models (PRMs) can generate step- or token-level feedback, but dense process rewards are costly to annotate and brittle across open-ended tool-use settings. Instead, tree rollouts already provide a form of implicit credit assignment by comparing continuations that share a prefix (Ji et al., 2025; Zhao et al., 2026). TRACE strengthens this mechanism without explicit process supervision by allocating more branches to prefixes whose alternative continuations are likely to diverge in outcome, turning sparse final rewards into sharper local comparison signals.

B Discussion

B.1 Allocation as an Orthogonal Layer

TRACE separates rollout acquisition from policy optimization. Tree-structured objectives such as TreeRPO (Yang et al., 2025b) and Tree-GRPO (Ji et al., 2025) define how an existing tree becomes gradients through branch comparisons, value targets, or group-relative advantages. TRACE operates upstream by deciding which anchors deserve additional descendants before the optimizer sees the tree. Thus the same policy objective can receive a contrast-poor or contrast-rich tree under the same rollout budget.

The predictor and TreePO instance used in our experiments are practical instantiations rather than primary technical commitments. TRACE only requires a predictor that ranks anchors by expected reward contrast and a tree-structured optimizer that consumes the resulting rollout tree. Better predictors should improve budget placement, but the framework is not tied to a specific predictor architecture or tree-policy objective.

B.2 Axes of Training-Time Sampling

Table 3 organizes training-time sampling methods by the allocation axes they make controllable. GRPO exposes only a flat sampling axis, PCL turns prompt admission into a learned decision, and TreePO exposes a tree substrate while leaving the tree acquisition policy random. The missing axis is not another optimizer choice, but a structural sampling decision inside the rollout itself. Once a trajectory has interacted with a search engine, a Python interpreter, or an API environment, its visited prefixes become decision states with heterogeneous downstream uncertainty. Some have already

Table 4: **Performance comparison on DeepScaler datasets.** We evaluate Qwen3-8B and Qwen3-14B across four in-distribution math reasoning datasets and three out-of-distribution scenarios. Our method consistently outperforms competitive baselines across all metrics. The best results are indicated in **bold**.

Model	Method	In-distribution						Out-of-distribution			
		AIME24	AMC23	MATH500	Minerva	Olympiad	Avg	MMLU-Pro	ARC-c	GPQA	Avg
Qwen3-8B	ReAct	52.3	87.0	89.8	37.8	58.6	65.1	68.5	95.3	52.4	72.1
	GRPO	63.6	91.0	92.3	40.2	62.8	70.0	72.7	95.3	55.9	74.6
	PCL	64.0	91.5	92.8	40.7	63.0	70.4	72.9	95.4	56.2	74.8
	TreePO	64.3	91.3	92.6	40.9	63.1	70.4	73.0	95.2	56.4	74.9
	Ours	63.9	91.2	93.4	41.2	65.8	**71.1**	73.4	95.7	56.9	**75.3**
Qwen3-14B	ReAct	61.5	89.4	91.8	40.0	62.3	69.0	75.1	96.2	59.2	76.8
	GRPO	65.6	94.3	94.2	45.2	68.4	73.5	75.9	96.1	59.4	77.1
	PCL	66.2	95.1	95.0	45.1	67.6	73.9	76.1	96.2	59.7	77.3
	TreePO	66.4	95.0	94.8	45.4	67.8	74.0	76.0	96.4	59.8	77.4
	Ours	66.1	94.7	94.9	47.3	71.5	**74.9**	76.5	96.5	60.4	**77.8**

Table 5: **Performance comparison on Multi-Hop Question Answering and Berkeley Function Calling Leaderboard (BFCL) benchmarks.** We evaluate Qwen3-8B and Qwen3-14B across four multi-hop reasoning datasets and four tool-calling scenarios. Our method consistently outperforms competitive baselines across all metrics. The best results are indicated in **bold**.

Model	Method	Multi-Hop QA					BFCL				
		Hotpot	2Wiki	Musiq	Bamb	Avg	Base	Miss-Func	Miss-Param	Long	Avg
Qwen3-8B	ReAct	41.9	37.6	18.4	44.7	35.7	36.7	21.5	31.1	22.3	28.0
	GRPO	53.2	48.7	27.4	64.7	48.5	58.5	31.0	45.8	38.7	43.5
	PCL	53.6	49.2	27.9	65.0	48.9	59.0	32.1	46.8	39.3	44.3
	TreePO	54.0	49.7	28.4	65.9	49.5	58.9	32.0	46.5	39.4	44.2
	Ours	55.7	50.9	29.5	66.3	**50.6**	61.2	34.4	48.8	40.4	**46.2**
Qwen3-14B	ReAct	46.4	44.2	22.8	55.3	42.2	51.1	23.6	35.6	22.8	33.6
	GRPO	55.3	52.7	30.1	66.8	51.2	70.6	32.6	40.9	36.4	46.1
	PCL	55.6	52.9	30.2	67.2	51.5	70.0	31.8	41.4	40.4	45.9
	TreePO	55.1	55.3	31.0	70.6	53.0	70.8	33.0	43.2	39.4	46.6
	Ours	57.8	53.1	33.7	71.3	**54.0**	72.4	34.8	44.6	40.2	**48.0**

collapsed into near-certain success or failure, while others remain poised at points where an alternative query, tool call, or reasoning continuation can alter the final reward. TRACE treats prompt roots and such prefixes as anchors in a single allocation geometry: a lightweight generalizable predictor scores where contrast is still latent, and the allocator decides whether to skip a root, widen it, or branch from an unresolved prefix.

B.3 Root-to-Prefix Predictor Generalization

Using one predictor for roots and prefixes is not merely an implementation shortcut. A prefix history is an information-refined prompt: it contains the current plan, the executed actions, and the environment observations that shape what remains possible. Root-level supervision is denser and anchors the predictor's global difficulty scale, while prefix-level supervision is sparser but closer to the decisions that create local contrast. The positive prefix-level correlations in Appendix C indicate that the predictor is not memorizing prompt identities. It learns a transferable history-conditioned notion of uncertainty, which is precisely the signal needed for targeted branching.

C Extended Experimental Results

C.1 Detailed Benchmark Results

Figure 4 shows training curves with respect to optimization steps, while Tables 4 and 5 summarize the final endpoint performance. Overall, TRACE enables more effective RLVR training under the same rollout-budget setting. Compared with GRPO, TRACE improves the average scores by 0.7–2.8 points across DeepScaler, Multi-Hop QA, and BFCL. Compared with random TreePO allocation, TRACE also consistently improves the reported averages, indicating that the gains come from learned allocation rather than from introduc-

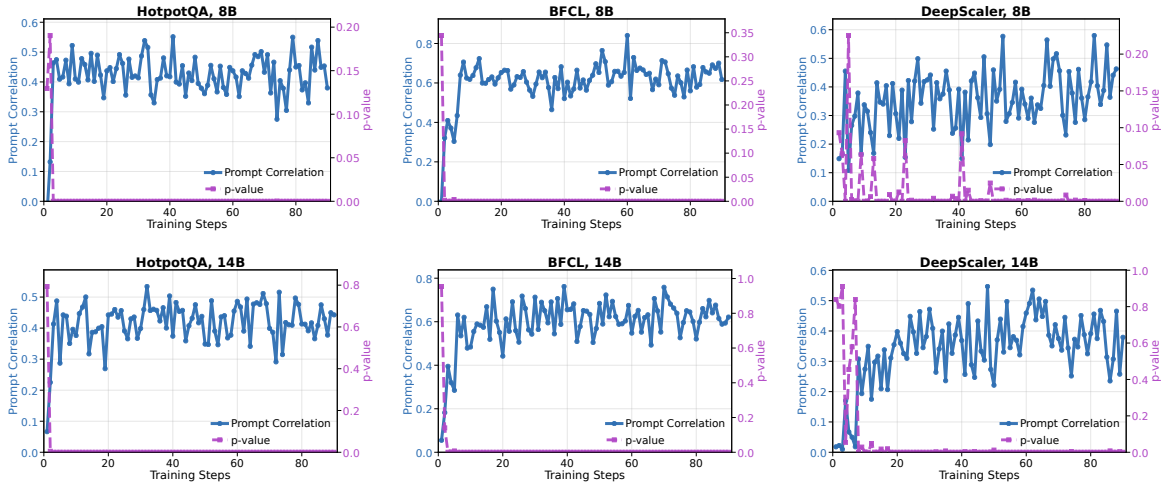


Figure 6: **Prompt-level prediction quality during training.** The panels show Spearman's rank correlation between predicted prompt difficulty and empirical success rate, with the associated significance shown on the right axis.

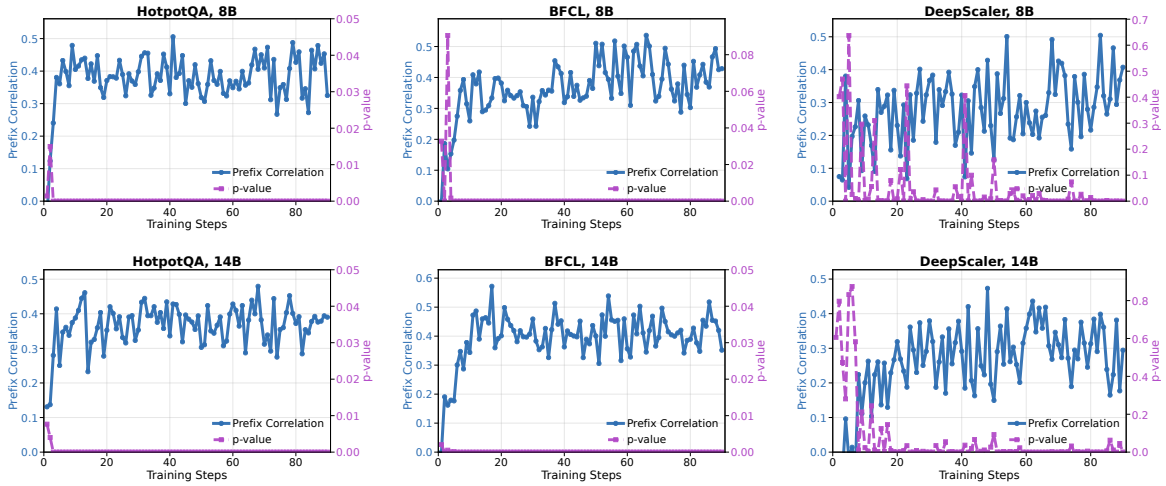


Figure 7: **Prefix-level prediction quality during training.** The panels show Spearman's rank correlation between predicted prefix difficulty and empirical continuation success, with the associated significance shown on the right axis.

ing a tree substrate alone. These improvements come with negligible predictor overhead, analyzed in Appendix C.5.

C.2 Effective Ratio

For the root or prefix stage $q \in \{\text{root}, \text{pref}\}$, we use the anchor set $\mathcal{A}_q$, descendant set $\mathcal{D}_h(b_h)$, and activation event $\mathsf{Act}(h, b_h)$ from Eq. (10). The effective ratio is the fraction of sampled terminal descendants attached to activated anchors:

$$\mathrm{Eff}_q(b) = \frac{\sum_{h \in \mathcal{A}_q} |\mathcal{D}_h(b_h)| \, \mathbb{I}\{\mathsf{Act}(h, b_h)\}}{\sum_{h \in \mathcal{A}_q} |\mathcal{D}_h(b_h)|}. \quad (21)$$

Higher values indicate that more sampled trajectories can produce non-degenerate group-relative or local contrast. In our diagnostic figures and ablation tables, we report the prompt-level instantiation because it is shared by flat and tree rollouts. For a training batch $\mathcal{B}_t$, let $R_i = \{r_{i,j}\}_{j=1}^{m_i}$ be the terminal rewards of all sampled descendants under prompt root x_i. We compute

$$\mathrm{Eff}_{\text{root}}(\mathcal{B}_t) = \frac{1}{|\mathcal{B}_t|} \sum_{x_i \in \mathcal{B}_t} \mathbb{I}\left[\mathrm{Var}(R_i) > 0\right]. \quad (22)$$

Here, $\mathbb{I}[\cdot]$ is the indicator function. This metric measures the fraction of prompt-root groups that contain both successful and failed outcomes. It is an end-to-end batch informativeness measure: root allocation and useful prefix expansion can both increase the chance that a prompt produces mixed

Table 6: **Llama-3.2-3B Multi-HopQA accuracy.** To test transfer beyond Qwen backbones, we report answer-match reward scores on four multi-hop QA benchmarks and their average.

Method	Hotpot	2Wiki	Musiq	Bamb	Avg
ReAct	6.7	7.3	2.9	6.6	5.9
GRPO	38.2	30.5	13.0	32.3	28.5
PCL	39.3	32.5	13.4	30.5	28.9
TreePO	39.7	33.6	13.6	29.9	29.2
Ours	41.0	35.3	15.2	37.7	**32.3**

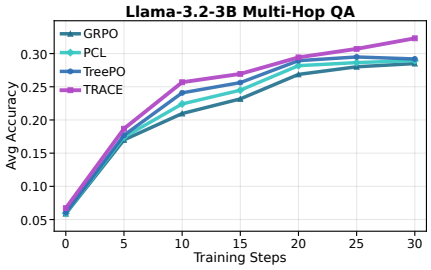


Figure 8: **Llama-3.2-3B Multi-HopQA average accuracy.** We report the four-dataset average over HotpotQA, 2WikiMultiHopQA, MuSiQue, and Bamboogle.

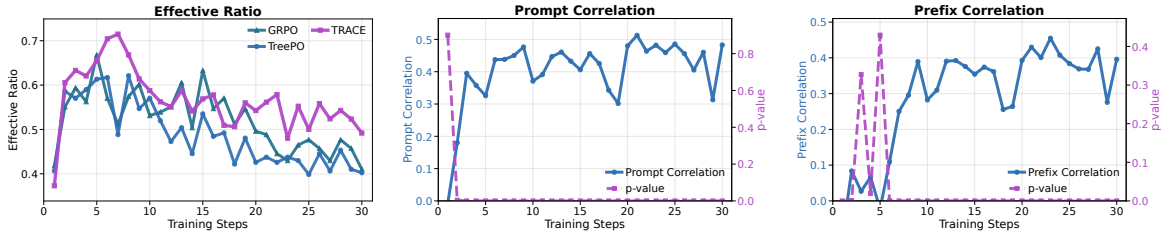


Figure 9: **Llama-3.2-3B allocation and predictor diagnostics.** The panels follow Figures 5–7, showing effective ratio, prompt-level correlation, and prefix-level correlation. Correlation panels show the predictor diagnostics.

outcomes.

Figure 5 shows that TRACE consistently produces more informative prompt-root groups than GRPO and random TreePO allocation. This metric isolates whether the prompts entering an update yield non-degenerate outcome variation, so higher values indicate that learned allocation supplies denser group-relative signal under the same rollout budget.

C.3 Spearman Correlation

To evaluate the quality of difficulty prediction, we report Spearman's rank correlation coefficient (Sedgwick, 2014). It measures the strength of the monotonic relationship between predicted and empirical difficulty rankings and is invariant to monotonic transformations, making it the right metric for allocation, where relative ordering matters more than calibration. For a root-level or prefix-level evaluation set $\mathcal{S}_q$, where $q \in \{\text{root}, \text{pref}\}$, let $R_\psi(y)$ and $R_{\widehat{V}}(y)$ be the ranks of the predictor score $\tilde{V}_\psi(y)$ from Eq. (18) and the tree-backed empirical target $\widehat{V}(y)$ from Eq. (19) within $\mathcal{S}_q$. We compute

$$\rho_q = \frac{\mathrm{cov}_{y\in\mathcal{S}_q}\big(R_\psi(y), R_{\widehat{V}}(y)\big)}{\mathrm{std}_{y\in\mathcal{S}_q}\big(R_\psi(y)\big)\,\mathrm{std}_{y\in\mathcal{S}_q}\big(R_{\widehat{V}}(y)\big)}. \tag{23}$$

We also report the corresponding p-value under the null hypothesis that $\tilde{V}_\psi(y)$ and $\widehat{V}(y)$ are independent over $y \in \mathcal{S}_q$, which assesses whether the observed rank agreement is statistically significant.

Figures 6 and 7 show that the predictor learns stable difficulty rankings at both levels. Prompt-level correlation rises first because roots provide the densest supervision. Prefix-level correlation remains positive as well, even though predictor training is dominated by prompt-level examples and uses only limited prefix-level supervision to expose the history format. This transfer is consistent with the root-to-prefix generalization view in Appendix B.3, suggesting that the predictor is not memorizing prompt IDs but naturally extending a root-trained difficulty model to serialized interaction histories, which is exactly the signal needed for targeted branching.

C.4 Applicability to Other Model Families

We further test TRACE on Llama-3.2-3B-Instruct to examine whether the allocation interface in Section 4 transfers beyond the Qwen3 family. Consistent with the Multi-Hop QA trend in Section 5.2, Table 6 shows that TRACE achieves the best average accuracy, improving over GRPO by 3.8 points and over random TreePO allocation by 3.1 points. The gains appear across HotpotQA, 2WikiMulti-

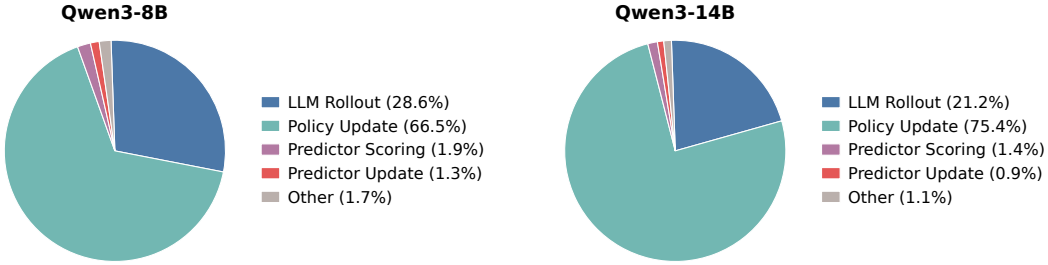
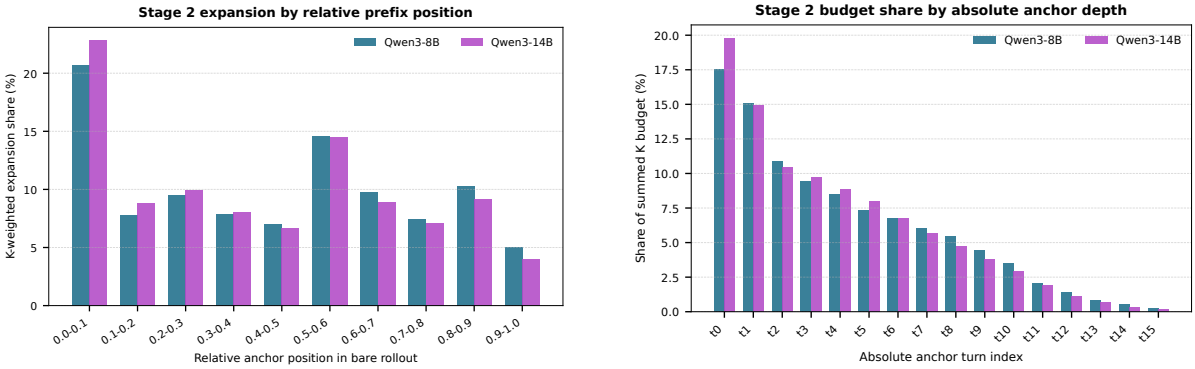


Figure 10: **Training time breakdown on HotpotQA.** We report average wall-clock time for TRACE with Qwen3-8B and Qwen3-14B. The current implementation uses an unoptimized predictor scoring path, while predictor parameter updates remain a small fraction of total runtime.



(a) Relative prefix position

(b) Absolute anchor depth

Figure 11: **Stage 2 allocation behavior on BFCL.** We average over the common late training window. The left panel bins anchors by relative position within their bare rollout and weights each anchor by the assigned continuation count $K_{i,j,t}$. The right panel reports the share of the total Stage 2 continuation budget assigned to each absolute anchor depth.

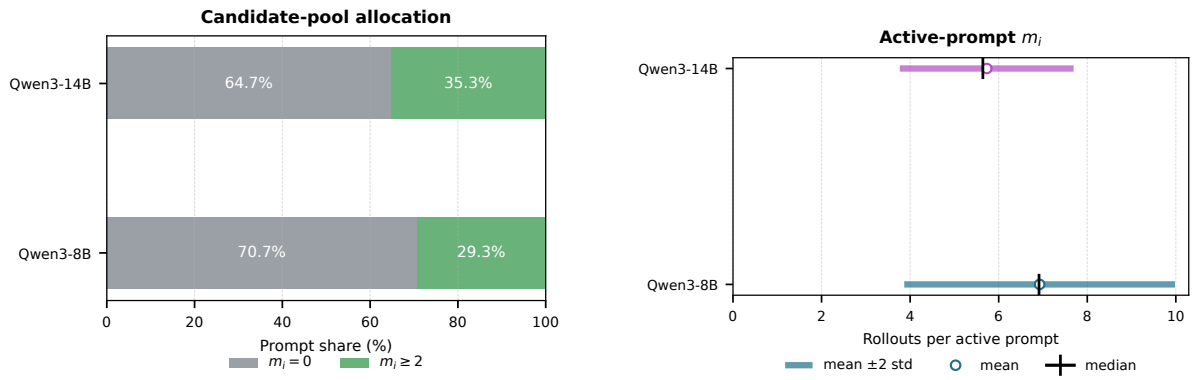
HopQA, MuSiQue, and Bamboogle, suggesting that the benefit is not tied to a single benchmark instance.

Figure 8 shows that the advantage emerges during training rather than from a stronger starting point. Figure 9 provides the corresponding allocation diagnostics, showing that TRACE maintains a higher effective ratio and that the predictor preserves useful prompt- and prefix-level rankings. Together, these results support the central design claim that TRACE is a portable root-prefix allocation layer rather than a policy-backbone-specific heuristic.

C.5 Computational Overhead

As representative, we profile the HotpotQA training runs to quantify where wall-clock time is spent. Figure 10 decomposes the runtime into LLM rollout, policy update, predictor scoring, predictor update, and residual overhead. The decomposition uses nonoverlapping critical path timers in which predictor scoring is subtracted from trajectory collection and predictor update is separated from trajectory transformation. This avoids double counting nested diagnostic timers.

The dominant cost is still policy optimization, which takes 66.5% of the Qwen3-8B runtime and 75.4% of the Qwen3-14B runtime. The learnable predictor update is small, accounting for 1.3% and 0.9% of total runtime. Predictor scoring during root and prefix allocation accounts for 1.9% and 1.4% of total runtime. Because TRACE uses the same Qwen3-0.6B predictor for both policy scales, total predictor overhead decreases from 3.2% on Qwen3-8B to 2.3% on Qwen3-14B, while the policy update becomes increasingly dominant. These results indicate that TRACE trades a bounded allocation cost for more informative rollouts, and that further engineering improvements can primarily target batched prefix scoring.



(a) Candidate-pool allocation

(b) Active-prompt root counts

Figure 12: **Stage 1 root allocation summary on BFCL.** Panel (a) shows the fraction of candidate prompts receiving $m_i = 0$ or $m_i \geq 2$; by construction TRACE uses $m_i \in \{0\} \cup \{2, 3, \ldots\}$, so $m_i = 1$ has zero mass. Panel (b) shows active-prompt m_i mean, median, and two-standard-deviation range over the same late training window. Exact per-integer counts within the active set require logging the full integer histogram of m_i.

C.6 Allocation Behavior

We inspect BFCL as a representative allocation trace because its longer function-calling episodes make turn-wise prefix allocation smoother than in shorter QA rollouts. Figure 11 shows two complementary views of Stage 2. The relative-position view normalizes each anchor by the length of its bare rollout, while the absolute-depth view reports the share of continuation budget assigned to each turn index. In Figure 11(a), the budget-weighted prefix position remains near the middle of the bare rollout, matching the half-trajectory continuation approximation used in the budget accounting of Appendix E.5. The absolute-depth view further shows that nontrivial budget remains across later prefixes up to depth 15, indicating that Stage 2 is not merely adding extra root-like samples.

Figure 12 summarizes Stage 1 root allocation over the candidate prompt pool. Most candidates receive $m_i = 0$ and are skipped, while active prompts receive multiple bare rollouts, typically around five to seven on BFCL. The current logs record exact inactive/active ratios and active-set quantiles; per-integer counts within the active set require logging the full integer histogram of m_i.

D Theoretical Proof

D.1 Proof of Proposition 1

Proof. Conditioned on $\mathcal{H}_t$, the m terminal rewards in $\bar{R}_{t,m}$ are independent Bernoulli variables with mean V_t^π. Hence

$$\mathbb{E}[\bar{R}_{t,m} \mid \mathcal{H}_t] = V_t^\pi, \tag{24}$$

so the Bayes-optimal predictor of group correctness is V_t^π. Its optimal risk is

$$\mathcal{E}^\star_{t,m} = \frac{1}{m}\mathbb{E}\big[V_t^\pi(1 - V_t^\pi)\big]. \tag{25}$$

Because $V_t^\pi = \mathbb{E}[V_{t+1}^\pi \mid \mathcal{H}_t]$, the law of total variance gives

$$\mathbb{E}\big[V_{t+1}^\pi(1 - V_{t+1}^\pi)\big] \leq \mathbb{E}\big[V_t^\pi(1 - V_t^\pi)\big]. \tag{26}$$

Therefore,

$$\mathcal{E}^\star_{t,m} - \mathcal{E}^\star_{t+1,m} = \frac{1}{m}\mathbb{E}\big[\operatorname{Var}(V_{t+1}^\pi \mid \mathcal{H}_t)\big] \geq 0. \tag{27}$$

The inequality is strict exactly when the next prefix changes the conditional success probability with positive probability under $\mathcal{H}_t$. Summing the one-step inequalities gives $\mathcal{E}^\star_{t,m} \leq \mathcal{E}^\star_{0,m}$ for every t. □

D.2 Proof of Proposition 2

Proof. Let $Z_t := V_t^\pi = \mathbb{E}_\pi\big[r(\mathcal{H}_T) \mid \mathcal{F}_t\big]$, where $\mathcal{F}_t := \sigma(\mathcal{H}_t)$. Since (Z_t) is a Doob martingale, we have $\mathbb{E}_\pi[Z_{s+1} \mid \mathcal{F}_s] = Z_s$ and therefore $\mathbb{E}_\pi[A_s^\pi \mid \mathcal{F}_s] = 0$ for $A_s^\pi := Z_{s+1} - Z_s$. Using

$$\begin{aligned} Z_{s+1}^2 - Z_s^2 &= (Z_{s+1} - Z_s)^2 \\ &\quad + 2Z_s(Z_{s+1} - Z_s), \end{aligned} \tag{28}$$

and summing from $s = t$ to $T - 1$, we obtain

$$Z_T^2 - Z_t^2 = \sum_{s=t}^{T-1}(A_s^\pi)^2 + 2\sum_{s=t}^{T-1} Z_s A_s^\pi. \tag{29}$$

Taking conditional expectation with respect to $\mathcal{F}_t$, the second term vanishes by the tower property:

$$\begin{aligned} \mathbb{E}_\pi[Z_s A_s^\pi \mid \mathcal{F}_t] &= \mathbb{E}_\pi\big[\mathbb{E}_\pi[Z_s A_s^\pi \mid \mathcal{F}_s] \mid \mathcal{F}_t\big] \\ &= \mathbb{E}_\pi\big[Z_s\, \mathbb{E}_\pi[A_s^\pi \mid \mathcal{F}_s] \mid \mathcal{F}_t\big] \\ &= 0. \end{aligned} \tag{30}$$

Hence

$$\mathbb{E}_\pi\left[[Z]_{t:T} \mid \mathcal{F}_t\right] = \mathbb{E}_\pi[Z_T^2 \mid \mathcal{F}_t] - Z_t^2. \quad (31)$$

Because the terminal reward is binary, $Z_T = r(\mathcal{H}_T) \in \{0,1\}$ and thus $Z_T^2 = Z_T$. Therefore,

$$\mathbb{E}_\pi[Z_T^2 \mid \mathcal{F}_t] = \mathbb{E}_\pi[Z_T \mid \mathcal{F}_t] = Z_t, \quad (32)$$

which yields

$$\mathbb{E}_\pi\left[[Z]_{t:T} \mid \mathcal{F}_t\right] = Z_t - Z_t^2 = V_t^\pi\big(1 - V_t^\pi\big). \quad (33)$$

This proves Eq. (9). □

D.3 Proof of Proposition 3

Proof. Fix an anchor a and a budget decision b. Let $\mathsf{Act}(a,b)$ denote the event that the sampled descendants below a contain at least one successful and one failed terminal history.

Step 1: outcome contrast is the activation event. For a root group of size m, activation means that the group is non-degenerate:

$$\Pr(\mathsf{Act}(a,m) \mid a) = 1 - p_a^m - (1-p_a)^m. \quad (34)$$

This is Eq. (12) after replacing p_a by the predictor. For a visited prefix occurrence with factual reward r, a new continuation matches the factual outcome with probability

$$q_a(r) = rp_a + (1-r)(1-p_a). \quad (35)$$

With k new continuations, activation means at least one opposite-outcome sibling appears:

$$\Pr(\mathsf{Act}(a,k) \mid a, r) = 1 - q_a(r)^k, \quad (36)$$

which is Eq. (16).

Step 2: common optimizers have zero local gradient without activation. For pairwise Bradley–Terry or DPO-style losses, an informative pair consists of two terminal histories $\mathcal{H}^+, \mathcal{H}^-$ below a with different rewards. Let

$$\ell_\theta(\mathcal{H} \mid a) = \log \frac{\pi_\theta(\mathcal{H} \mid a)}{\pi_{\text{ref}}(\mathcal{H} \mid a)},$$
$$\Delta_\theta = \ell_\theta(\mathcal{H}^+ \mid a) - \ell_\theta(\mathcal{H}^- \mid a).$$

Then the Bradley–Terry gradient is

$$\begin{aligned}\nabla_\theta \log \sigma(\beta\Delta_\theta) &= \beta\sigma(-\beta\Delta_\theta)\nabla_\theta\Delta_\theta \\ &= \beta\sigma(-\beta\Delta_\theta)\big[\nabla_\theta \log \pi_\theta(\mathcal{H}^+ \mid a) \\ &\quad -\nabla_\theta \log \pi_\theta(\mathcal{H}^- \mid a)\big]\end{aligned} \quad (37)$$

If all sampled rewards below a are identical, no such success–failure pair exists and the local pairwise contribution is zero.

For group-relative updates, let $\mathcal{H}_1, \ldots, \mathcal{H}_m$ be sampled descendants with rewards $R_s \in \{0,1\}$ and mean $\bar{R}$. Ignoring clipping and constants, the local group-relative gradient contribution is

$$G_{\text{grp}}(a) = \sum_{s=1}^{m} (R_s - \bar{R}) \nabla_\theta \log \pi_\theta(\mathcal{H}_s \mid a). \quad (38)$$

For any vectors $g_1, \ldots, g_m$, the algebraic identity

$$\sum_{s=1}^{m} (R_s - \bar{R}) g_s = \frac{1}{m} \sum_{u<v} (R_u - R_v)(g_u - g_v), \quad (39)$$

holds. Taking $g_s = \nabla_\theta \log \pi_\theta(\mathcal{H}_s \mid a)$ shows that $G_{\text{grp}}(a)$ is a sum of success–failure score differences; if the group is all-success or all-failure, every coefficient vanishes.

Now let $\widetilde{G}_a(b)$ denote the local gradient shape that the chosen optimizer would produce from the available success–failure contrasts below anchor a, and let $G_a(b)$ be the realized local gradient contribution. The pairwise and group-relative arguments above show that no such contribution is available when no opposite-outcome descendants are observed. Tree-aware optimizers only redistribute such leaf-level contrast through the tree, so they also have zero local contrast contribution below a when $\mathsf{Act}(a,b)$ fails. Hence, for all optimizer classes considered here,

$$G_a(b) = \mathbb{I}\{\mathsf{Act}(a,b)\}\widetilde{G}_a(b). \quad (40)$$

Step 3: activation factors the expected squared gradient norm. Taking squared norms and conditional expectation in Eq. (40) gives

$$\begin{aligned}\mathbb{E}\big[\|G_a(b)\|^2 \mid a\big] &= \Pr(\mathsf{Act}(a,b) \mid a) \\ &\cdot \mathbb{E}\Big[\|\widetilde{G}_a(b)\|^2 \mid \mathsf{Act}(a,b), a\Big].\end{aligned} \quad (41)$$

By definition, $V_a(b)$ denotes the appropriate activation probability: $\Pr(\mathsf{Act}(a,b) \mid a)$ for roots and $\Pr(\mathsf{Act}(a,b) \mid a, r)$ for prefix anchors with factual reward r. If the conditional squared gradient norm lies in $[c_-, c_+]$ across candidate anchors, then the expected squared local gradient norm is bounded between $c_- V_a(b)$ and $c_+ V_a(b)$.

Step 4: dominance of local gradient energy. For a stage $q \in \{\text{root}, \text{pref}\}$, let h index the candidate

anchors in that stage and let b_h be the budget assigned to anchor h. Step 3 gives the total expected squared local gradient norm in the form

$$\mathcal{E}_q(b) = \sum_h V_h(b_h) C_h(b_h),$$

where $C_h(b_h)$ is the conditional squared gradient scale after activation. This factor is optimizer- and anchor-dependent, while $V_h(b_h)$ is exactly the activation probability controlled by rollout allocation. Under the normalized-gradient condition in Proposition 3, $C_h(b_h) = c_q$ for every anchor in stage q. TRACE therefore optimizes the gradient-energy component

$$\mathcal{J}_q(b) := \sum_h V_h(b_h), \tag{42}$$

where V_h is Eq. (12) for $q = \text{root}$ and Eq. (16) for $q = \text{pref}$. TRACE solves the resulting combinatorial optimization over the feasible set $\mathcal{F}_q$. If $b_q^\star \in \arg\max_{b \in \mathcal{F}_q} \mathcal{J}_q(b)$ and $b_q^u \in \mathcal{F}_q$ is a uniform baseline, then

$$\mathcal{E}_q(b_q^\star) = c_q \mathcal{J}_q(b_q^\star) \geq c_q \mathcal{J}_q(b_q^u) = \mathcal{E}_q(b_q^u). \tag{43}$$

This proves the first line of Eq. (11) for both $q = \text{root}$ and $q = \text{pref}$.

For the combined tree update, write $G_r := G_{\text{root}}(b_{\text{root}})$ and $G_p := G_{\text{pref}}(b_{\text{pref}})$. Expanding the squared norm gives

$$\mathbb{E}\|G_r + G_p\|^2 = \mathcal{E}_{\text{root}}(b_{\text{root}}) + \mathcal{E}_{\text{pref}}(b_{\text{pref}}) + 2\mathbb{E}\langle G_r, G_p \rangle.$$

The two energy terms are no smaller under TRACE by the stage-wise result above. The cross-term condition in Proposition 3 states that TRACE does not reduce $\mathbb{E}\langle G_r, G_p \rangle$ relative to uniform, so the full expected squared gradient norm is also no smaller. Strictness holds for any stage whose uniform allocation is not an optimizer of $\mathcal{J}_q$. This proves Proposition 3. □

E Experimental Details

E.1 Tasks

E.1.1 Mathematical Reasoning

Training dataset. We train on the DeepScaler (Luo et al., 2025) corpus, a collection of 40,315 competition-level mathematics problems. We use the public DeepScaleR preview dataset hosted at `https://huggingface.co/datasets/agentica-org/DeepScaleR-Preview-Dataset`. Each rollout is executed by a tool-augmented agent with access to a Python interpreter for calculation and verification.

Evaluation benchmarks. We evaluate in-distribution mathematical reasoning on AIME24, AMC23, MATH500 (Lightman et al., 2024), MinervaMath (Lewkowycz et al., 2022), and OlympiadBench (He et al., 2024a), using the math evaluation sets hosted at `https://huggingface.co/datasets/math-ai`. Per-dataset scores use the same sampling multiplicities as our evaluation scripts: AIME24 is reported as avg@32, AMC23 as avg@16, and MATH500, MinervaMath, and OlympiadBench as avg@4. To test out-of-distribution generalization, we additionally evaluate on MMLU-Pro (Wang et al., 2024), ARC-Challenge (Clark et al., 2018), and GPQA-diamond (Rein et al., 2023), reported as avg@1, avg@4, and avg@8 respectively. The "Avg" columns in the tables are arithmetic averages across the corresponding benchmark columns.

Reward function. We use a binary verifier reward: a trajectory receives reward 1 if the final extracted answer matches the ground truth and 0 otherwise. The same reward interface is used for base-model evaluation and RL checkpoints.

E.1.2 Multi-Hop Question Answering

Training dataset. We train on HotpotQA (Yang et al., 2018), using the HuggingFace version hosted at `https://huggingface.co/datasets/hotpotqa/hotpot_qa`. The agent follows a ReAct-style search-and-answer loop, issuing search queries to a local E5 retrieval server (Wang et al., 2022) built over a Wikipedia dump (Karpukhin et al., 2020). This keeps retrieval deterministic and avoids external search-engine drift during RL training.

Evaluation benchmarks. We evaluate on HotpotQA, 2WikiMultiHopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022), and Bamboogle (Press et al., 2023). The additional held-out sets are prepared from the FlashRAG benchmark repository at `https://huggingface.co/datasets/RUC-NLPIR/FlashRAG_datasets`, with canonical sources at `https://github.com/Alab-NII/2wikimultihop`, `https://github.com/StonyBrookNLP/musique`, and `https://huggingface.co/datasets/`

chiayewken/bamboogle. These datasets stress different forms of multi-hop evidence aggregation, from entity chaining to compositional question decomposition. Multi-Hop QA evaluation uses one sampled trajectory per question, so each per-dataset score is avg@1. The reported "Avg" is the arithmetic average over the four QA benchmarks when per-checkpoint evaluation is available.

Reward function. We use the implemented HotpotQA-style search reward as both the training reward and evaluation score. The evaluator first extracts a concise final answer and normalizes it following SQuAD/HotpotQA conventions. An exact match receives reward 1; otherwise, the evaluator computes token-level F1 against all reference answers, treats predictions with F1 at least 0.3 as partially correct, and assigns reward equal to the F1 score. Predictions below this threshold receive reward 0.

E.1.3 Function Calling

Dataset. We use the multi-turn split of BFCL v4 (Patil et al., 2025), rather than the full BFCL v4 suite, which also contains many single-turn, live, memory, and web-search categories. We use the four agentic multi-turn subsets from https://huggingface.co/datasets/gorilla-llm/Berkeley-Function-Calling-Leaderboard: base, long-context, missing-function, and missing-parameter. We split these multi-turn examples into an 80% training portion and a 20% held-out test portion. The agent observes API specifications, calls functions over multiple turns, and receives environment feedback after each call.

Evaluation protocol. We follow the official BFCL evaluation protocol and report success rates over the base, missing-function, missing-parameter, and long-context subsets. Validation uses four sampled trajectories per example, so subset scores are reported as avg@4. The reported "Avg" is the arithmetic average over the four multi-turn BFCL subsets. A trajectory is correct only when the executed API sequence and final task outcome satisfy the evaluator.

E.2 Models

All main experiments use Qwen3-8B and Qwen3-14B (Yang et al., 2025a) policy backbones, loaded from the official Qwen model family. We additionally run Multi-Hop QA experiments with Llama-3.2-3B-Instruct as an extra backbone in Appendix C. The ReAct baseline evaluates the corresponding base or instruction-tuned model in the same multi-turn agent scaffold without RL fine-tuning. TRACE uses a lightweight Qwen3-0.6B (Yang et al., 2025a) model as the prefix predictor. The relevant model repositories are:

- Qwen3-8B: https://huggingface.co/Qwen/Qwen3-8B;
- Qwen3-14B: https://huggingface.co/Qwen/Qwen3-14B;
- Llama-3.2-3B-Instruct: https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct;
- Qwen3-0.6B predictor backbone: https://huggingface.co/Qwen/Qwen3-0.6B.

E.3 Training Details

All RL experiments are implemented in the rLLM/verl (Sheng et al., 2024; Tan et al., 2025) stack with asynchronous vLLM rollout workers. Unless otherwise stated, actor optimization uses Adam with learning rate $1e{-}6$, no KL penalty, sequence-mean token aggregation, gradient clipping at 1.0, and Clip-Higher ranges $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.28$. We use FSDP with parameter and optimizer offloading for the actor and reference model. Main training runs use two nodes with eight GPUs per node.

The environment horizon and sequence lengths are task-specific. For Multi-Hop QA, we use maximum prompt length 2048, maximum response length 6144, and at most 5 agent turns. For DeepScaler, we use maximum prompt length 2048, maximum response length 14336, and at most 5 agent turns. For BFCL, we use maximum prompt length 12800, maximum response length 8192, and at most 40 agent turns to accommodate long multi-turn tool-use traces.

Rollout sampling follows each benchmark's baseline setting. Multi-Hop QA uses temperature 1.0 and top-$p = 1.0$ during training. DeepScaler uses temperature 0.6 and top-$p = 0.95$, matching the math evaluation style. BFCL uses temperature 0.9 and top-$p = 1.0$.

E.4 Policy Optimizers and TreePO Instances

TRACE is optimizer-agnostic and outputs tree-structured rollout data that can be consumed by different tree-aware RL objectives. In the flat baseline, GRPO normalizes terminal rewards within

a response group and removes the value network used in PPO. Let $\mathcal{B}_{\mathrm{G}}$ denote $x \sim \mathcal{D}$ and $\{y_i\}_{i=1}^{k} \sim \pi_{\boldsymbol{\theta}_{\mathrm{old}}}(\cdot \mid x)$, and write $\bar{\rho} = \mathrm{clip}(\rho, 1-\epsilon_{\mathrm{low}}, 1+\epsilon_{\mathrm{high}})$. GRPO optimizes

$$\mathcal{J}_{\mathrm{GRPO}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{B}_{\mathrm{G}}}\Bigg[\frac{1}{k}\sum_{i=1}^{k}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\Bigg(\min\Big(\rho_{i,t}\hat{A}_i, \bar{\rho}_{i,t}\hat{A}_i\Big) - \beta D_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}}\|\pi_{\mathrm{ref}})\Bigg)\Bigg], \tag{44}$$

where $\rho_{i,t} = \frac{\pi_{\boldsymbol{\theta}}(y_{i,t}|x, y_{i,<t})}{\pi_{\boldsymbol{\theta}_{\mathrm{old}}}(y_{i,t}|x, y_{i,<t})}$ and

$$\hat{A}_i = \frac{r_i - \mathrm{mean}(\{r_j\}_{j=1}^{k})}{\mathrm{std}(\{r_j\}_{j=1}^{k})}. \tag{45}$$

For stable training, we use two concrete tree-structured optimizer instances. DeepScaler and HotpotQA use a TreeRPO-style update, while BFCL uses Tree-GRPO. In TreeRPO, each leaf ℓ receives a verifiable outcome reward $R(\ell) = \phi(P_\ell)$ for its full root-to-leaf path P_ℓ, and each internal node backs up the average child reward

$$R(v) = \frac{1}{|\mathcal{C}(v)|}\sum_{c \in \mathcal{C}(v)} R(c). \tag{46}$$

For every parent p, its child set $\mathcal{C}(p)$ forms a local group, with child advantage

$$\hat{A}_c^{\mathrm{TR}} = \frac{R(c) - \mu_p}{\sigma_p}, \qquad \mu_p = \frac{1}{|\mathcal{C}(p)|}\sum_{c' \in \mathcal{C}(p)} R(c'). \tag{47}$$

Let $\mathcal{B}_{\mathrm{TR}}$ denote sampling a prompt and its tree from $\pi_{\boldsymbol{\theta}_{\mathrm{old}}}$. TreeRPO consumes prefix-level groups with the same clipped GRPO-style update:

$$\mathcal{J}_{\mathrm{TR}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{B}_{\mathrm{TR}}}\Bigg[\frac{1}{|\mathcal{E}_{\mathrm{tr}}|}\sum_{(p,c)\in\mathcal{E}_{\mathrm{tr}}}\frac{1}{|o_c|}\sum_{t=1}^{|o_c|}\Bigg(\min\Big(\rho_{c,t}\hat{A}_c^{\mathrm{TR}}, \bar{\rho}_{c,t}\hat{A}_c^{\mathrm{TR}}\Big) - \beta D_{\mathrm{KL}}\Bigg)\Bigg]. \tag{48}$$

BFCL uses Tree-GRPO, where each tree node is a complete thought-action-observation step. Tree-GRPO combines an intra-tree group, formed by rollouts sharing the same initialized tree, and an inter-tree group, formed by all rollouts for the prompt:

$$\hat{A}_i^{\mathrm{intra/inter}} = \frac{R(H_i) - \mu_{\mathrm{intra/inter}}}{\sigma_{\mathrm{intra/inter}}}, \qquad \hat{A}_i^{\mathrm{TG}} = \hat{A}_i^{\mathrm{intra}} + \hat{A}_i^{\mathrm{inter}}. \tag{49}$$

Let $\mathcal{B}_{\mathrm{TG}}$ denote sampling $x \sim \mathcal{D}$ and $\mathcal{G}(x) \sim \pi_{\boldsymbol{\theta}_{\mathrm{old}}}$. The resulting objective is

$$\mathcal{J}_{\mathrm{TG}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{B}_{\mathrm{TG}}}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|H_i|}\sum_{t=1}^{|H_i|}\Bigg(\min\Big(\rho_{i,t}\hat{A}_i^{\mathrm{TG}}, \bar{\rho}_{i,t}\hat{A}_i^{\mathrm{TG}}\Big) - \beta D_{\mathrm{KL}}\Bigg)\Bigg]. \tag{50}$$

In all cases, TreePO denotes the same tree-aware optimizer family with random root and prefix allocation, while TRACE changes only the allocation policy that decides where tree branches are collected.

E.5 Rollout Construction and Budget Accounting

All methods draw from the same task-dependent candidate pool, but they consume it differently. For the fixed-size prompt baselines, Multi-Hop QA selects 256 prompts from 512 candidates, while DeepScaler and BFCL select 128 prompts from 256 candidates in the matched 8B settings. GRPO selects this subset uniformly, and PCL selects it with its difficulty predictor. Each selected prompt then receives 8 flat rollouts. TreePO uses the same selected-prompt count and total budget, but collects a random tree by sampling 4 bare rollouts per selected prompt and randomly choosing two branch points in each bare rollout.

TRACE does not first commit to a fixed number of selected prompts. It solves Stage 1 over the full candidate pool and assigns integer root counts m_i, where $m_i = 0$ skips a candidate and $m_i > 0$ determines how many bare rollouts are generated. Figure 12 shows that the resulting active-prompt fraction does not exceed the fixed-size prompt baselines, so the comparison is not advantaged by exposing TRACE to more distinct prompts. In Multi-Hop QA we use $M = 1024$ root rollout budget units and $N = 2$ continuation budget units per active root; in DeepScaler and BFCL we use $M = 512$ and $N = 2$. After the m_i bare rollouts in Eq. (5) finish, each trajectory induces visited

prefix histories $\mathcal{H}_{i,j,t}$ for $1 \leq t < T_{i,j}$. Stage 2 allocates $m_i N$ continuation slots over these prefix occurrences and samples the additional suffixes in Eq. (6). We use one expansion round in the main experiments, so newly generated suffixes are evaluated as terminal branches and are not recursively expanded.

For budget accounting, we count a root rollout as one trajectory unit and approximate a continuation from an internal prefix as one half trajectory. Thus, ignoring rare fallback cases, TRACE uses approximately $M(1 + N/2)$ trajectory units and is matched to a flat baseline with B prompts and G rollouts per prompt by choosing (M, N) such that $M(1 + N/2) \approx BG$. If no valid prefix candidate is available for a prompt, we convert the local continuation budget into additional root rollouts. In implementation, Stage 2 can be triggered per prompt as soon as that prompt's bare rollouts finish, which avoids a batch-wide straggler barrier while preserving the logical two-stage semantics.

E.6 Prefix Predictor Supervision and Implementation

The prefix predictor $\tilde{V}_\psi$ is implemented as a Qwen3-0.6B critic model, extending the lightweight value-model setup used by PCL (Gao et al., 2025; Qu et al., 2026) from prompt roots to serialized prefix histories. For each queried anchor, the input is the serialized prompt and the interaction history up to that prompt root or prefix. The predictor outputs a scalar success score in $[0, 1]$, which is used by both Stage 1 root allocation and Stage 2 prefix allocation. For multiple prefixes under the same prompt, prefix-cache reuse amortizes the shared prompt and early-history computation, keeping the additional scoring overhead small.

Following Eqs. (19)–(20), the predictor is updated online after each rollout step by fitting scalar tree-backed success targets with an MSE value-model objective. Each update uses the current step's prompt-root examples together with a small stream of prefix examples, with prefix samples accounting for 6% of the predictor batch. Predictor learning rates are $3\mathrm{e}{-5}$ for Multi-Hop QA and BFCL, and $1.5\mathrm{e}{-5}$ for DeepScaler. We train for two predictor epochs per rollout on Multi-Hop QA and BFCL, and four epochs per rollout on DeepScaler.

E.7 Code Availability

Upon acceptance, we will release the TRACE implementation, including rollout allocation, tree construction, and training scripts for the experiments in this paper. The code will be distributed under the MIT license.

E.8 Artifact Use and Intended Use

All existing artifacts in this work are used in ways consistent with their stated research purposes. Public datasets, model weights, and software frameworks listed in Appendix E are employed only for training and evaluating language-model agents on reasoning, retrieval, and tool-use benchmarks. We do not repurpose them for unrelated commercial, surveillance, or high-risk deployment settings.

The artifacts we release, including TRACE code, training scripts, and any derived preprocessing utilities, are intended for academic research and reproducibility. They should be used under the same research-only scope as the upstream datasets and model licenses that govern the original data and checkpoints. In particular, derivatives of data accessed for research should not be reused outside research contexts unless permitted by the corresponding license terms.

E.9 Use of AI Assistants

We used AI assistants to polish the language of this manuscript.

F Data Examples

This section shows representative examples from the actual rLLM data and agent interfaces used by our training scripts. We include the system message seen by the agent, including the tool schemas injected by `ToolAgent`.

DeepScaler & Mathematics Benchmarks example

System prompt:

```
You are a math assistant that can write
python to solve math problems.
# Tools
You may call one or more functions to assist
with the user query.
You are provided with function signatures
within <tools></tools> XML tags:
<tools>
[
{
"type": "function",
"function": {
"name": "python",
```

"description": "Execute Python code in a sandboxed environment. Returns results and standard output/error.",
"parameters": {
"type": "object",
"properties": {
"code": {
"type": "string",
"description": "Execute Python code in a sandboxed environment. Returns results and standard output/error."
},
"timeout": {
"type": "integer",
"description": "Maximum execution time in seconds before timing out",
"default": 12
}
},
"required": [
"code"
]
}
}
}
]
</tools>
For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

Prompt:
The operation $\otimes$ is defined for all nonzero numbers by $a \otimes b = \frac{a^{2}}{b}$.
Determine $[(1 \otimes 2) \otimes 3] - [1 \otimes (2 \otimes 3)]$.

Ground-Truth Answer:
-\frac{2}{3}

Multi-Hop QA example

System prompt:
You are a helpful AI assistant that can search for information to answer questions accurately.
When answering questions:
1. Use the available search tools to find relevant and reliable information
2. Synthesize information from multiple sources when needed
3. Provide accurate and comprehensive answers based on your search results
4. Always put your final answer in \boxed{} format
For example:
- If the answer is "American", write: \boxed{American}
- If the answer is "yes", write: \boxed{yes}
- If the answer is a year like "1985", write: \boxed{1985}
Remember to search thoroughly and provide your final answer clearly within the \boxed{} format.
Tools
You may call one or more functions to assist with the user query.
You are provided with function signatures within <tools></tools> XML tags:
<tools>
[
{
"type": "function",
"function": {
"name": "local_search",
"description": "Search for information using a dense retrieval server with Wikipedia corpus",
"parameters": {
"type": "object",
"properties": {
"query": {
"type": "string",
"description": "Search query to retrieve relevant documents"
},
"top_k": {
"type": "integer",
"description": "Number of results to return (default: 10)",
"minimum": 1,
"maximum": 50
}
},
"required": [
"query"
]
}
}
}
]
</tools>
For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

Prompt:
Were Scott Derrickson and Ed Wood of the same nationality?

Ground-Truth Answer:
yes

BFCL function-calling example

System prompt:
Tools
You may call one or more functions to assist with the user query.
You are provided with function signatures within <tools></tools> XML tags:
<tools>
Important: Always use only the latest tool list provided, ignoring any functions mentioned in previous messages.
{"name": "add_to_watchlist", "description": "This tool belongs to the trading system,

which allows
users to trade stocks, manage their account,
and view stock information. Tool description:
Add a
stock to the watchlist.", "parameters":
{"type": "dict", "properties": {"stock":
{"type": "string",
"description": "the stock symbol to add to
the watchlist. "}}, "required": ["stock"]},
"response":
{"type": "dict", "properties": {"watchlist":
{"type": "array", "description": "the
watchlist.",
"items": {"type": "string"}}}}}
{"name": "cancel_order", "description": "This
tool belongs to the trading system, which
allows users
to trade stocks, manage their account, and
view stock information. Tool description:
Cancel an
order.", "parameters": {"type": "dict",
"properties": {"order_id": {"type":
"integer",
"description": "ID of the order to cancel.
"}}, "required": ["order_id"]}, "response":
{"type":
"dict", "properties": {"order_id": {"type":
"integer", "description": "ID of the
cancelled order."},
"status": {"type": "string", "description":
"New status of the order after cancellation
attempt."}}}}
{"name": "filter_stocks_by_price",
"description": "This tool belongs to the
trading system, which
allows users to trade stocks, manage their
account, and view stock information. Tool
description:
Filter stocks based on a price range.",
"parameters": {"type": "dict", "properties":
{"stocks":
{"type": "array", "items": {"type": "string"},
"description": "List of stock symbols to
filter."},
"min_price": {"type": "float", "description":
"Minimum stock price."}, "max_price":
{"type":
"float", "description": "Maximum stock price.
"}}, "required": ["stocks", "min_price",
"max_price"]}, "response": {"type": "dict",
"properties": {"filtered_stocks": {"type":
"array",
"description": "Filtered list of stock
symbols within the price range.", "items":
{"type":
"string"}}}}}
{"name": "fund_account", "description": "This
tool belongs to the trading system, which
allows users
to trade stocks, manage their account, and
view stock information. Tool description:
Fund the
account with the specified amount.",
"parameters": {"type": "dict", "properties":
{"amount":
{"type": "float", "description": "Amount to
fund the account with. "}}, "required":
["amount"]},
"response": {"type": "dict", "properties":
{"status": {"type": "string", "description":
"Status of
the funding operation."}, "new_balance":
{"type": "float", "description": "Updated
account balance
after funding."}}}}
{"name": "get_account_info", "description":
"This tool belongs to the trading system,
which allows
users to trade stocks, manage their account,
and view stock information. Tool description:
Get
account information.", "parameters": {"type":
"dict", "properties": {}, "required": []},
"response":
{"type": "dict", "properties": {"account_id":
{"type": "integer", "description": "ID of the
account."}, "balance": {"type": "float",
"description": "Current balance of the
account."},
"binding_card": {"type": "integer",
"description": "Card number associated with
the account."}}}}
{"name": "get_available_stocks",
"description": "This tool belongs to the
trading system, which
allows users to trade stocks, manage their
account, and view stock information. Tool
description:
Get a list of stock symbols in the given
sector.", "parameters": {"type": "dict",
"properties":
{"sector": {"type": "string", "description":
"The sector to retrieve stocks from (e.g.,
'Technology'). "}}, "required": ["sector"]},
"response": {"type": "dict", "properties":
{"stock_list": {"type": "array",
"description": "List of stock symbols in the
specified sector.",
"items": {"type": "string"}}}}}
{"name": "get_current_time", "description":
"This tool belongs to the trading system,
which allows
users to trade stocks, manage their account,
and view stock information. Tool description:
Get the
current time.", "parameters": {"type":
"dict", "properties": {}, "required": []},
"response":
{"type": "dict", "properties":
{"current_time": {"type": "string",
"description": "Current time in
HH:MM AM/PM format."}}}}
{"name": "get_order_details", "description":
"This tool belongs to the trading system,
which allows
users to trade stocks, manage their account,
and view stock information. Tool description:
Get the
details of an order.", "parameters": {"type":
"dict", "properties": {"order_id": {"type":
"integer",
"description": "ID of the order. "}},
"required": ["order_id"]}, "response":
{"type": "dict",
"properties": {"id": {"type": "integer",
"description": "ID of the order."},
"order_type": {"type":

"string", "description": "Type of the order."}, "symbol": {"type": "string", "description": "Symbol of the stock in the order."}, "price": {"type": "float", "description": "Price at which the order was placed."}, "amount": {"type": "integer", "description": "Number of shares in the order."}, "status": {"type": "string", "description": "Current status of the order. [Enum]: [\"Open\", \"Pending\", \"Completed\", \"Cancelled\"]"}}}}
{"name": "get_order_history", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Get the stock order ID history.", "parameters": {"type": "dict", "properties": {}, "required": []}, "response": {"type": "dict", "properties": {"order_history": {"type": "array", "description": "List of orders ID in the order history.", "items": {"type": "integer"}}}}}
{"name": "get_stock_info", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Get the details of a stock.", "parameters": {"type": "dict", "properties": {"symbol": {"type": "string", "description": "Symbol that uniquely identifies the stock. "}}, "required": ["symbol"]}, "response": {"type": "dict", "properties": {"price": {"type": "float", "description": "Current price of the stock."}, "percent_change": {"type": "float", "description": "Percentage change in stock price."}, "volume": {"type": "float", "description": "Trading volume of the stock."}, "MA(5)": {"type": "float", "description": "5-day Moving Average of the stock."}, "MA(20)": {"type": "float", "description": "20-day Moving Average of the stock."}}}}
{"name": "get_symbol_by_name", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Get the symbol of a stock by company name.", "parameters": {"type": "dict", "properties": {"name": {"type": "string", "description": "Name of the company. "}}, "required": ["name"]}, "response": {"type": "dict", "properties": {"symbol": {"type": "string", "description": "Symbol of the stock or \"Stock not found\" if not available."}}}}
{"name": "get_transaction_history", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Get the transaction history within a specified date range.", "parameters": {"type": "dict", "properties": {"start_date": {"type": "string", "description": "Start date for the history (format: 'YYYY-MM-DD')."}, "default": "None"}, "end_date": {"type": "string", "description": "End date for the history (format: 'YYYY-MM-DD'). ", "default": "None"}}, "required": []}, "response": {"type": "dict", "properties": {"transaction_history": {"type": "array", "description": "List of transactions within the specified date range.", "items": {"type": "dict", "properties": {"type": {"type": "string", "description": "Type of transaction. [Enum]: [\"deposit\", \"withdrawal\"]"}, "amount": {"type": "float", "description": "Amount involved in the transaction."}, "timestamp": {"type": "string", "description": "Timestamp of the transaction, formatted as 'YYYY-MM-DD HH:MM:SS'."}}}}}}}
{"name": "get_watchlist", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Get the watchlist.", "parameters": {"type": "dict", "properties": {}, "required": []}, "response": {"type": "dict", "properties": {"watchlist": {"type": "array", "description": "List of stock symbols in the watchlist.", "items": {"type": "string"}}}}}
{"name": "notify_price_change", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Notify if there is a significant price change in the stocks.", "parameters": {"type": "dict", "properties": {"stocks": {"type": "array", "items": {"type": "string"}, "description": "List of stock symbols to check."}, "threshold": {"type": "float", "description": "Percentage change threshold to trigger a notification. "}}, "required": ["stocks", "threshold"]}, "response": {"type": "dict", "properties": {"notification": {"type": "string", "description": "Notification message about the price changes."}}}}

{"name": "place_order", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Place an order.", "parameters": {"type": "dict", "properties": {"order_type": {"type": "string", "description": "Type of the order (Buy/Sell)."}, "symbol": {"type": "string", "description": "Symbol of the stock to trade."}, "price": {"type": "float", "description": "Price at which to place the order."}, "amount": {"type": "integer", "description": "Number of shares to trade. "}}, "required": ["order_type", "symbol", "price", "amount"]}, "response": {"type": "dict", "properties": {"order_id": {"type": "integer", "description": "ID of the newly placed order."}, "order_type": {"type": "string", "description": "Type of the order (Buy/Sell)."}, "status": {"type": "string", "description": "Initial status of the order."}, "price": {"type": "float", "description": "Price at which the order was placed."}, "amount": {"type": "integer", "description": "Number of shares in the order."}}}}
{"name": "remove_stock_from_watchlist", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Remove a stock from the watchlist.", "parameters": {"type": "dict", "properties": {"symbol": {"type": "string", "description": "Symbol of the stock to remove. "}}, "required": ["symbol"]}, "response": {"type": "dict", "properties": {"status": {"type": "string", "description": "Status of the removal operation."}}}}
{"name": "trading_get_login_status", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Get the login status.", "parameters": {"type": "dict", "properties": {}, "required": []}, "response": {"type": "dict", "properties": {"status": {"type": "boolean", "description": "Login status."}}}}
{"name": "trading_login", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Handle user login.", "parameters": {"type": "dict", "properties": {"username": {"type": "string", "description": "Username for authentication."}, "password": {"type": "string", "description": "Password for authentication. "}}, "required": ["username", "password"]}, "response": {"type": "dict", "properties": {"status": {"type": "string", "description": "Login status message."}}}}
{"name": "trading_logout", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Handle user logout for trading system.", "parameters": {"type": "dict", "properties": {}, "required": []}, "response": {"type": "dict", "properties": {"status": {"type": "string", "description": "Logout status message."}}}}
{"name": "withdraw_funds", "description": "This tool belongs to the trading system, which allows users to trade stocks, manage their account, and view stock information. Tool description: Withdraw funds from the account balance.", "parameters": {"type": "dict", "properties": {"amount": {"type": "float", "description": "Amount to withdraw from the account. "}}, "required": ["amount"]}, "response": {"type": "dict", "properties": {"status": {"type": "string", "description": "Status of the transaction."}, "new_balance": {"type": "float", "description": "Updated account balance after the transaction."}}}}
</tools>
For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

Prompt:

Turn 1: Can you provide the latest trading details for Quasar Ltd.?
Turn 2: Furthermore, I'd like to delve into the technology sector. Would you be able to compile a comprehensive list of stock symbols pertinent to this industry and add all of them to my watchlist

Ground-Truth API sequence:

```
[
[
"get_stock_info(symbol='QUAS')"
],
[
"get_available_stocks('Technology')",
```

```
"add_to_watchlist(stock='AAPL')",
"add_to_watchlist(stock='GOOG')",
"add_to_watchlist(stock='MSFT')"
]
]
```