PACT: PRIVILEGED TRACE CO-TRAINING FOR MULTI-TURN TOOL-USE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-turn tool-use agents must reason, call tools, and adapt to observations across several interaction turns. Post-training such agents is challenging, as reinforcement learning often suffers from sparse rewards and weak credit assignment despite matching the prompt-only inference setting, while supervised fine-tuning on expert traces provides dense process supervision but can over-constrain the model to fixed trajectories. To tackle this, we propose **PACT**, a **P**rivileged tr**A**ce **C**o-**T**raining framework for multi-turn tool-use agents. The key idea is to use expert traces only as training-time optimization signals rather than rollout-time hints. PACT keeps rollout generation prompt-only, then uses expert traces to guide optimization through two complementary signals: a trace-conditioned RL surrogate that evaluates prompt-only rollouts under expert-trace context, and a component-aware SFT loss that provides annealed process supervision over reasoning and tool-call components. To reduce over-reliance on the training-only trace context, PACT incorporates prompt-only anchoring with standard prompt-only RL updates. Experiments on FTRL, BFCL, and ToolHop show that PACT consistently outperforms strong SFT- and RL-based baselines. Further analyses support the complementarity of the two training objectives and the benefits of component-aware supervision and prompt-only anchoring. These results highlight the effectiveness of privileged trace co-training for multi-turn tool-use learning.

1 INTRODUCTION

Tool-use extends LLMs beyond static text generation by enabling interaction with external resources, including search engines (Nakano et al., 2021; Zheng et al., 2025; Song et al., 2025; Jin et al., 2025), computational tools (Schick et al., 2023; Gou et al., 2024; Liao et al., 2024), and user-defined APIs (Li et al., 2023; Patil et al., 2024; Xu et al., 2026; Prabhakar et al., 2026). Many practical tool-use tasks are multi-turn, requiring an agent to reason about the current state, construct tool-calls through tool selection and argument specification, incorporate returned observations, and decide whether to continue or produce a final answer (Barres et al., 2025; Wang et al., 2025; 2026; Qu et al., 2026). Post-training such agents is therefore challenging because they require learning from final-task outcomes while also handling a multi-turn decision process.

Reinforcement Learning (RL) is a natural paradigm, which optimizes the policy from prompt-only rollouts using rewards based on task success or tool-use behavior (Shao et al., 2024; Yu et al., 2026; Qian et al., 2026; Pang et al., 2025). This matches inference-time behavior, where no expert trace is available. However, multi-turn tool-use rewards are often sparse or weakly localized, providing limited feedback on which intermediate decisions lead to success or failure (Wei et al., 2025; Qu et al., 2026). And it may struggle to discover successful multi-turn tool-use trajectories. Expert traces provide complementary process supervision through successful tool-use trajectories (Qin et al., 2024; Song et al., 2024; Chen et al., 2023). However, Supervised Fine-tuning (SFT) on these traces learns from fixed targets and can over-constrain the model to a single reference trajectory, even when different tool-use trajectories could solve the same problem (Chen et al., 2025b). Moreover, SFT alone does not explicitly assign credit to intermediate reasoning and tool-use decisions based on final task success (Chen et al., 2025a).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

These limitations motivate methods that combine expert traces with RL, but existing combinations do not directly address this setting. Hint-based methods provide expert solutions as additional rollout context, introducing a mismatch between hint-conditioned training rollouts and prompt-only inference (Liu et al., 2025). Joint SFT-RL methods add imitation losses to RL, but are mainly designed for single-turn reasoning traces rather than long interactions with tool-calls and observations (Chen et al., 2025a; Yuan et al., 2025; Zhang et al., 2026). Taken together, these approaches either change the rollout conditioning or apply imitation objectives that are not specifically tailored to the structure of multi-turn tool-use behavior. We therefore ask *how expert traces can guide joint SFT–RL training for multi-turn tool-use agents while preserving prompt-only rollout*.



Figure 1: PACT keeps rollouts prompt-only while using expert traces only as privileged optimization signals, improving multi-turn tool-use performance. FTRL reports Solve-F1.

We answer this question with **PACT**, a **P**rivileged tr**A**ce **C**o-**T**raining framework for multi-turn tool-use agents, as shown in Figure 1. Instead of using expert traces to condition rollouts or treating them as full imitation targets, PACT separates prompt-only rollout generation from trace-based optimization. Within this framework, expert traces guide joint SFT–RL training by providing privileged context for RL updates and supervision targets for component-aware SFT. Concretely, PACT introduces two trace-based signals. First, a trace-conditioned RL surrogate computes the policy likelihoods of prompt-only rollouts under expert-trace context, while retaining their original advantages for policy updates. Second, a component-aware SFT loss supervises selected reasoning prefixes and complete tool-calls instead of the full trace. With separate annealing schedules for different trace components, it provides early process guidance while gradually relaxing direct imitation as training progresses. We also provide a conceptual latent-trace view to motivate the joint objective and interpret the two roles of expert traces in PACT (Appendix E). To reduce over-reliance on the privileged trace context, PACT further incorporates prompt-only anchoring, routing a subset of prompts to standard RL without trace-based loss in each optimization step.

Our contributions are summarized below.

- We propose PACT, a privileged trace co-training framework for multi-turn tool-use agents. PACT separates prompt-only rollout generation from expert-trace-guided optimization.
- We introduce two core designs in PACT: trace-conditioned RL and component-aware SFT. The RL surrogate uses expert traces as privileged context to guide trace-conditioned RL updates, while the SFT supervises selected reasoning prefixes and complete tool-call spans with component-aware annealing.
- Experiments on multiple multi-turn tool-use benchmarks show that PACT consistently outperforms strong SFT- and RL-based baselines. Ablation studies support the empirical complementarity of trace-conditioned RL and component-aware SFT, as well as the benefits of component-aware supervision schedules.

2 Related Work

Tool-use agents. Tool-use agents have been developed through prompting and instruction tuning to interleave reasoning with external actions, including ReAct, Toolformer, and ToolLLM (Yao et al., 2022; Schick et al., 2023; Qin et al., 2024). Recent work further applies RL to tool-use learning through interactive environments, reward design, and credit assignment. ToolStar and FTRL construct environments for tool-use RL (Dong et al., 2025; Ye et al., 2026), while Agent Lightning decouples agent execution from RL optimization (Luo et al., 2025). ToolRL (Qian et al., 2026) instead designs rewards for tool selection and application, and MatchTIR (Qu et al., 2026) further

assigns fine-grained credit to tool-integrated reasoning steps. In contrast, PACT does not modify the reward function or environment. It introduces privileged trace co-training for multi-turn tool-use agents, using expert traces to jointly guide SFT and RL while preserving prompt-only rollout generation.

Combining SFT and RL. In general reasoning settings, recent work combines expert supervision with RL through different mechanisms. UFT uses partial-solution hints to bridge imitation and exploration (Liu et al., 2025), BRIDGE coordinates SFT and RL through bilevel optimization (Chen et al., 2025a), and MIFO mitigates forgetting between the two training paradigms (Yuan et al., 2025). CHORD combines on-policy RL with an auxiliary SFT objective controlled by a global coefficient and token-wise weights (Zhang et al., 2026). More recently, OPSD trains on student rollouts using token-level supervision from a teacher conditioned on privileged information (Penaloza et al., 2026; Zhao et al., 2026; Yang et al., 2026). Multi-turn tool use, however, calls for supervision that accounts for the distinct roles of reasoning, tool-calls, and observations. PACT keeps rollout generation prompt-only and uses expert traces only during optimization, as privileged context for RL likelihood evaluation and process supervision over model-controllable trace components.

3 PRELIMINARIES

3.1 TASK FORMULATION

We consider multi-turn tool-use tasks, where an LLM agent is given a user prompt x and a set of available tools $\mathcal{T}$, and interacts with an external tool environment over multiple turns. At each turn t, the agent generates a reasoning step u_t and a structured tool-call a_t, which specifies an invoked tool from $\mathcal{T}$ and its arguments. The environment executes the tool-call and returns an observation o_t, which is appended to the interaction context. The process terminates when the agent produces a final answer y or reaches a predefined maximum number of tool-use turns.

A rollout trajectory with T tool-use turns can be written as

$$\tau = \big(u_1, a_1, o_1, u_2, \ldots, u_T, a_T, o_T, y\big), \tag{1}$$

where u_t, a_t, and y are generated by the agent, while o_t is returned by the environment and used only as context for subsequent decisions. The rollout receives a scalar reward $R(x, \tau)$ that evaluates the final outcome and, when available, intermediate tool-use behavior.

In this work, we additionally give each training prompt x_i an expert tool-use trace

$$h_i^* = \big(u_{i,1}^*, a_{i,1}^*, o_{i,1}^*, \ldots, u_{i,T_i^*}^*, a_{i,T_i^*}^*, o_{i,T_i^*}^*\big). \tag{2}$$

The detailed construction of expert traces is described in Appendix C. To avoid directly exposing the expert's final answer, we exclude its explicit final-answer y_i^* span from h_i^*.

3.2 POST-TRAINING WITH SFT AND RL

Given the rollout trajectories and expert traces defined above, two representative approaches for agentic LLM post-training are SFT and RL. Let π_θ denote the policy parameterized by θ. SFT learns from expert traces, while RL optimizes the policy using rewards from rollouts.

Supervised Fine-Tuning. Given an expert trace h^* for prompt x, SFT maximizes the likelihood of the expert's model-generated tokens. Let $\mathcal{D}_{\text{exp}}$ denote the expert-trace dataset, and let $\mathcal{M}(h^*)$ denote the model-generated token positions in the serialized trace. The SFT loss is given by

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(x,h^*)\sim\mathcal{D}_{\text{exp}}}\left[\sum_{n\in\mathcal{M}(h^*)} \log \pi_\theta\left(h_n^* \mid x, h_{<n}^*\right)\right]. \tag{3}$$

Tool observations remain in the conditioning history $h_{<n}^*$ but are excluded from the loss. This objective provides dense process supervision, but it trains the policy to follow fixed expert traces rather than optimizing the outcomes of its own rollouts. It can also over-constrain learning when multiple valid tool-use trajectories exist.

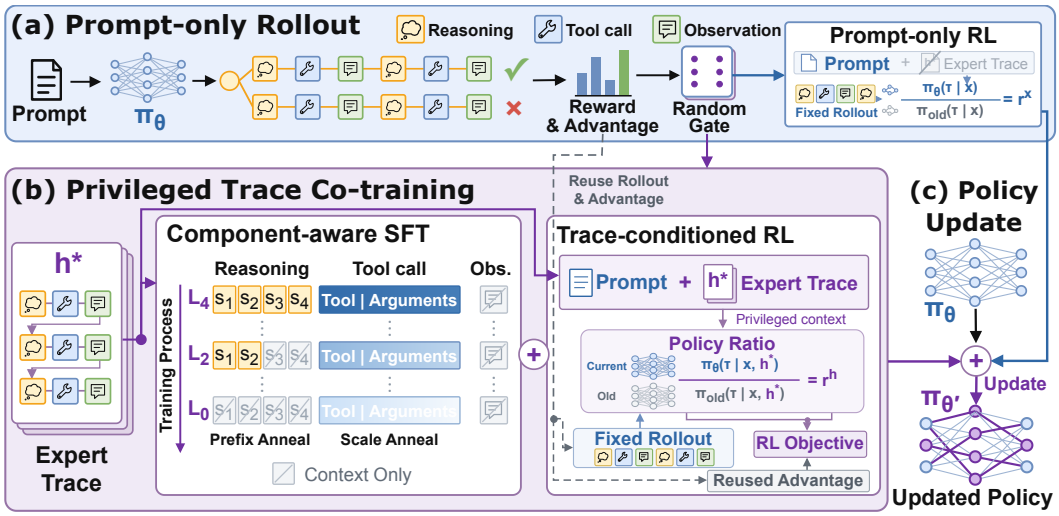

Figure 2: **Overview of PACT.** (a) Prompt-only rollouts provide rewards and advantages. (b) Expert traces provide supervision targets for component-aware SFT and privileged context for trace-conditioned RL. (c) The combined objectives update the policy.

Group-based Reinforcement Learning. One of the representative RL methods is Group Relative Policy Optimization (GRPO) (Shao et al., 2024), which estimates relative advantages from multiple rollouts sampled for the same prompt. For each prompt x_i, an old policy π_{old} samples a group of K rollouts $\tau_{i,k} \sim \pi_{\text{old}}(\cdot \mid x_i)$, $k = 1, \ldots, K$. Each rollout receives a reward $R(x_i, \tau_{i,k})$, and the rewards are normalized within the group to obtain an advantage $A_{i,k}$, or its token-level variant $A_{i,k,n}$, where n indexes positions of tokens generated in the rollout.

For a generated token $\tau_{i,k,n}$, the prompt-only importance ratio is

$$r^{x}_{i,k,n}(\theta) = \frac{\pi_\theta(\tau_{i,k,n} \mid x_i, \tau_{i,k,<n})}{\pi_{\text{old}}(\tau_{i,k,n} \mid x_i, \tau_{i,k,<n})}. \tag{4}$$

The clipped GRPO loss with KL regularization is

$$\mathcal{L}^{x}_{\text{RL}} = -\mathbb{E}_{i,k,n}\left[\min\left\{r^{x}_{i,k,n}A_{i,k,n}, \text{clip}\left(r^{x}_{i,k,n}, 1-\epsilon, 1+\epsilon\right) A_{i,k,n}\right\} - \beta_{\text{KL}} D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})\right], \tag{5}$$

where ϵ is the clipping threshold, β_{KL} is the KL coefficient, and π_{ref} is a fixed reference policy. The expectation is over prompts, sampled rollouts, and model-generated tokens. This prompt-only formulation uses the task prompt and interaction history, without access to expert traces.

4 Method

In this section, we introduce PACT as illustrated in Figure 2.

4.1 Prompt-only Rollout Generation

For each training prompt x_i, PACT samples K tool-use rollouts from the old policy without access to the expert trace h^*_i:

$$\tau_{i,k} \sim \pi_{\text{old}}(\cdot \mid x_i), \qquad k = 1, \ldots, K. \tag{6}$$

At each turn, the policy conditions on the task prompt and the accumulated interaction history, matching the inference-time input setting. Unlike hint-guided methods that provide partial expert solutions as rollout context (Liu et al., 2025), PACT uses expert traces only after rollout collection. Rewards and advantages are computed from the prompt-only rollouts, while expert traces provide privileged context for trace-conditioned RL and supervision targets for component-aware SFT.

4.2 TRACE-CONDITIONED RL SURROGATE

After rollout generation, PACT uses expert traces as privileged context for policy optimization. While rollout rewards and advantages determine which generated actions to reinforce, expert traces provide additional context for computing the token-level policy ratios used in the update.

For prompt x_i with expert trace h_i^*, we define the privileged context as $c_i^h = [x_i; h_i^*]$ (see Appendix A.1 for details). We evaluate the same sampled rollout under both the current and old policies using this context, while keeping its rewards and advantages unchanged. Let $\mathcal{M}_{i,k}$ denote the model-generated token positions in rollout $\tau_{i,k}$. For $n \in \mathcal{M}_{i,k}$, the trace-conditioned policy ratio is

$$r^h_{i,k,n}(\theta) = \frac{\pi_\theta(\tau_{i,k,n} \mid c_i^h, \tau_{i,k,<n})}{\pi_{\text{old}}(\tau_{i,k,n} \mid c_i^h, \tau_{i,k,<n})}. \tag{7}$$

Using the token-level advantages $A_{i,k,n}$ (Qu et al., 2026) (Appendix B), we optimize the following clipped trace-conditioned RL surrogate:

$$\begin{aligned}\mathcal{L}^h_{\text{RL}}(i) = -\mathbb{E}_{k,\, n\in\mathcal{M}_{i,k}}\Big[\min\Big\{ r^h_{i,k,n} A_{i,k,n},\, \text{clip}\big(r^h_{i,k,n}, 1-\epsilon, 1+\epsilon\big)\, A_{i,k,n} \Big\} \\ - \beta_{\text{KL}} D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \Big].\end{aligned} \tag{8}$$

This objective evaluates prompt-only rollouts under expert-trace context while retaining their original advantages. We treat the clipped policy objective as an auxiliary surrogate, rather than an importance correction between the sampling and evaluation contexts.

4.3 COMPONENT-AWARE SFT

Trace-conditioned RL uses expert traces as conditioning context to guide the policy update, but does not directly use them as supervision targets. We therefore add an auxiliary component-aware SFT loss that provides direct process supervision from expert traces. We supervise model-controllable components, *reasoning tokens* and *tool-call tokens*, while keeping tool observations only as context. Reasoning tokens guide intermediate decisions, but full reasoning imitation may over-constrain the policy. Tool-call tokens teach valid invocation, but exact teacher actions may represent only one of several valid choices. These differences motivate separate annealing strategies: we progressively shorten the supervised reasoning prefix within each assistant turn and lower the weight assigned to tool-call tokens while keeping their supervised spans complete.

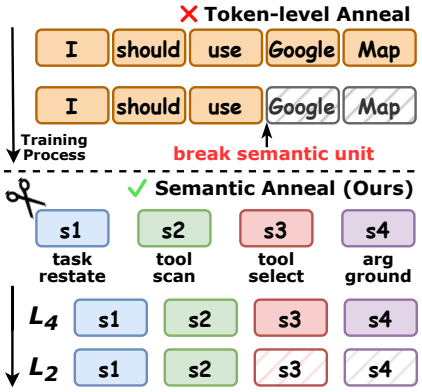

Figure 3: **Semantic reasoning-prefix annealing.** As training progresses, the supervised reasoning prefix is gradually shortened at semantic boundaries.

Component-aware SFT loss. For prompt x_i, let $h^*_{i,n}$ denote the n-th token in its expert trace. At training step s, we supervise the active reasoning prefixes and complete tool-call spans, with token positions indexed by $\mathcal{R}_i(s)$ and $\mathcal{A}_i$, respectively. The corresponding binary supervision masks are

$$m^r_{i,n}(s) = \mathbf{1}[n \in \mathcal{R}_i(s)], \qquad m^a_{i,n} = \mathbf{1}[n \in \mathcal{A}_i]. \tag{9}$$

We normalize the reasoning and tool-call losses separately and apply the tool-call weight after normalization:

$$\mathcal{L}_{\text{SFT}}(s) = -\sum_{i,n}\left[\frac{m^r_{i,n}(s)}{\sum_{j,\ell} m^r_{j,\ell}(s) + \epsilon_{\text{SFT}}} + \lambda_a(s)\frac{m^a_{i,n}}{\sum_{j,\ell} m^a_{j,\ell} + \epsilon_{\text{SFT}}}\right] \log \pi_\theta(h^*_{i,n} \mid x_i, h^*_{i,<n}). \tag{10}$$

Here, ϵ_{SFT} is a small constant for numerical stability. A component with no selected tokens contributes zero.

Annealed Reasoning Supervision. For reasoning, we supervise prefixes within each assistant turn to provide intermediate guidance without forcing imitation of the entire expert reasoning process. Instead of annealing by raw token count, we construct prefix levels over semantic reasoning segments, as illustrated in Figure 3. Token-level annealing may end supervision in the middle of a coherent reasoning unit. In contrast, segment-level annealing keeps the supervised prefixes aligned with semantic boundaries while gradually reducing supervision from longer to shorter prefixes, following $L_4 \to L_3 \to \cdots \to L_0$. At L_0, no reasoning tokens receive direct SFT supervision, although they remain in the teacher-forced context. Details of trace construction and the exact prefix-level definitions are provided in Appendix C.

Annealed Tool-call Supervision. For tool-calls, we supervise complete spans so that tool names and all argument tokens receive direct supervision. Since an expert tool-call may be only one of several valid actions, we retain these spans throughout training while gradually reducing their imitation weight:

$$\lambda_a(s) = \lambda_a^{\text{start}} - \Delta\lambda_a \min\Big(\frac{s}{S}, 1\Big), \tag{11}$$

where $\Delta\lambda_a = \lambda_a^{\text{start}} - \lambda_a^{\text{end}}$, S is the annealing horizon, and $\lambda_a^{\text{start}} > \lambda_a^{\text{end}}$.

A larger early weight provides stronger supervision for complete tool-calls, while a smaller later weight relaxes imitation of the expert's specific tool choices and arguments.

4.4 Overall Objective with Prompt-only Anchoring

For prompt x_i, PACT combines the trace-conditioned RL surrogate with component-aware SFT to form the trace co-training objective:

$$\mathcal{L}_i^{\text{trace}} = \mathcal{L}_{\text{RL}}^{h}(i) + \beta_{\text{SFT}}\mathcal{L}_{\text{SFT}}(i; s), \tag{12}$$

where $\mathcal{L}_{\text{SFT}}(i; s)$ denotes the SFT contribution associated with the expert trace for x_i, and β_{SFT} controls its weight.

Expert traces play two related roles in this joint objective: they provide direct supervision for learning the expert process and privileged context for reward-based updates on self-generated rollouts. Appendix E develops a conceptual latent-trace interpretation of these roles, while Section 5.3 evaluates their empirical complementarity.

Although rollout generation is prompt-only, the trace-conditioned RL surrogate evaluates policy ratios under privileged trace context. To reduce over-reliance on this training-only context, PACT also includes prompt-only anchoring. For each prompt x_i, we sample a gate $z_i \sim \text{Bernoulli}(q)$, shared by all K rollouts, where q is the prompt-only anchoring probability.

The final per-prompt objective selects between trace co-training and prompt-only RL:

$$\mathcal{L}_i = (1 - z_i)\mathcal{L}_i^{\text{trace}} + z_i\mathcal{L}_{\text{RL}}^{x}(i). \tag{13}$$

When $z_i = 0$, prompt x_i adopts the trace co-training loss in Eq. 12. When $z_i = 1$, it leverages only the standard prompt-only RL loss $\mathcal{L}_{\text{RL}}^{x}(i)$, without trace-conditioned likelihood evaluation or SFT supervision. This retains trace-guided co-training while preserving a direct optimization path under the prompt-only condition used at inference time.

5 Experiments

5.1 Experiment Settings

Models and datasets. We conduct experiments on Qwen3-1.7B and Qwen3-4B (Yang et al., 2025). We use prompts from the released FTRL (Ye et al., 2026) training split for model training and expert-trace construction. For the main experiments, expert traces are primarily generated with Qwen3-235B, then verified and segmented as described in Appendix C. During training, we use a batch size of 256 and a maximum of 10 interaction turns. For PACT, we set the SFT scale $\beta_{\text{SFT}} = 0.1$. We use a prompt-only anchoring probability of $q = 0.4$ for the main results. The objective, trace-component, and SFT-schedule ablations use $q = 0$ to isolate the effects of trace co-training. Additional training and evaluation details are provided in Appendix A.

Table 1: Main results on FTRL, BFCL, and ToolHop. AC denotes Answer Correctness. BFCL Avg averages Multi-Turn, Search, and Memory; the final Avg is the equal-weight mean of FTRL Solve-F1, BFCL Avg, and ToolHop AC.

Methods	FTRL			BFCL				ToolHop	Avg
	Solve-R	Solve-P	Solve-F1	Multi-Turn	Search	Memory	Avg	AC	
Qwen3-1.7B									
Vanilla	21.66	20.78	19.72	13.00	1.00	8.60	7.53	25.63	17.63
SFT	22.36	17.89	17.66	13.38	2.50	8.60	8.16	23.32	16.38
GRPO	21.57	17.37	18.13	11.88	3.50	9.25	8.21	25.13	17.16
FTRL	22.78	22.06	21.06	12.62	6.00	11.18	9.93	26.43	19.14
ToolRL	22.79	17.93	17.99	13.30	2.50	8.39	8.06	25.83	17.29
CHORD	22.68	19.14	18.50	12.62	3.50	6.45	7.52	27.14	17.72
OPSD	25.24	20.60	21.15	12.62	3.00	9.46	8.36	25.53	18.35
MatchTIR	24.12	21.50	21.43	14.00	4.50	9.03	9.18	32.26	20.96
SFT→MatchTIR	23.39	19.64	19.72	13.63	5.50	11.18	10.10	31.06	20.29
Ours	**28.33**	**26.94**	**22.93**	**14.88**	**7.00**	**13.63**	**11.84**	**34.47**	**23.08**
Qwen3-4B									
Vanilla	32.14	31.66	28.60	23.50	12.00	16.34	17.28	31.86	25.91
SFT	28.19	29.32	24.67	22.62	4.00	12.90	13.17	30.25	22.70
GRPO	33.12	31.80	30.84	22.00	9.00	15.48	15.49	34.47	26.93
FTRL	31.98	33.24	31.34	22.38	9.00	16.77	16.05	43.02	30.14
ToolRL	32.59	28.98	28.28	23.00	11.00	17.42	17.14	34.27	26.56
CHORD	32.69	29.84	29.88	22.00	12.50	15.48	16.66	37.69	28.08
OPSD	33.61	25.36	27.61	22.50	12.50	13.55	16.18	43.82	29.20
MatchTIR	36.10	30.60	31.60	24.38	17.50	14.62	18.83	47.54	32.66
SFT→MatchTIR	37.17	30.40	32.03	23.62	17.50	13.76	18.29	47.34	32.55
Ours	**42.41**	**34.69**	**36.60**	**27.50**	**18.00**	**18.28**	**21.26**	**49.55**	**35.80**

Baselines and metrics. We compare our method against several representative baselines, including the vanilla model, SFT, GRPO (Shao et al., 2024), FTRL (Ye et al., 2026), ToolRL (Qian et al., 2026), CHORD (Zhang et al., 2026), OPSD (Penaloza et al., 2026), MatchTIR (Qu et al., 2026), and SFT→MatchTIR. SFT, CHORD, OPSD, and SFT→MatchTIR share the same verified expert-trace pool as PACT; the remaining trained baselines do not use full expert traces as conditioning context or SFT targets (Appendix C.1). We evaluate all methods on three tool-use benchmarks: FTRL (Ye et al., 2026), BFCL (Patil et al., 2025), and ToolHop (Ye et al., 2025). We report solve rate (Solve-R), solve precision (Solve-P), and Solve F1 for FTRL, accuracy for BFCL, and Answer Correctness (AC) for ToolHop. Metric definitions are provided in Appendix A.

5.2 Main Results

Table 1 reports the main results on FTRL, BFCL, and ToolHop. PACT achieves the highest overall average score on both model sizes, improving over MatchTIR from 20.96 to 23.08 on Qwen3-1.7B and from 32.66 to 35.80 on Qwen3-4B. On FTRL, PACT improves Solve-F1 over the best baseline for each model size: from 21.43 to 22.93 over MatchTIR on Qwen3-1.7B, and from 32.03 to 36.60 over SFT→MatchTIR on Qwen3-4B. It also achieves the highest scores on all three reported BFCL subtasks and improves ToolHop AC from 32.26 to 34.47 and from 47.54 to 49.55 over MatchTIR on the two model sizes, respectively. PACT also consistently outperforms OPSD, improving the overall average from 18.35 to 23.08 on Qwen3-1.7B and from 29.20 to 35.80 on Qwen3-4B.

These results support our main design choice. Compared with SFT and SFT→MatchTIR, PACT incorporates expert traces into the RL update and progressively relaxes direct imitation through component-aware SFT. Compared with prompt-only RL baselines such as FTRL and MatchTIR, PACT provides additional guidance through expert-trace context and selective process supervision. The consistent gains show that keeping rollouts prompt-only while using traces only during optimization is effective for multi-turn tool-use learning.

Additional experiments provide further evidence for these results. PACT maintains its advantage over MatchTIR when both methods are trained for 48 updates on Qwen3-1.7B (Appendix F.1). PACT remains effective across different teacher models, with consistent gains over SFT→MatchTIR (Appendix F.2). Offline trace-construction and model-training costs are reported in Appendix F.3.

Table 2: Ablation on training objectives using Qwen3-4B on FTRL and BFCL, with prompt-only anchoring disabled ($q = 0$). Both combined objectives use the same component-aware SFT loss. BFCL Avg averages Multi-Turn, Search, and Memory accuracy.

	FTRL			BFCL
Objective	**Solve-R**	**Solve-P**	**Solve-F1**	**Avg**
Vanilla	32.14	31.66	28.60	17.28
$\mathcal{L}_{\mathrm{RL}}^{x}$	36.10	30.60	31.60	18.83
$\mathcal{L}_{\mathrm{RL}}^{h}$	35.29	31.42	31.85	19.07
$\mathcal{L}_{\mathrm{RL}}^{x}+\beta_{\mathrm{SFT}}\mathcal{L}_{\mathrm{SFT}}$	36.36	28.46	30.18	19.38
$\mathcal{L}_{\mathrm{RL}}^{h}+\beta_{\mathrm{SFT}}\mathcal{L}_{\mathrm{SFT}}$	**40.45**	**32.94**	**33.21**	**20.44**

5.3 Complementarity of Trace-Conditioned RL and SFT

To assess whether trace-conditioned RL and component-aware SFT provide complementary benefits, we compare prompt-only and trace-conditioned RL, each with and without component-aware SFT, on FTRL and BFCL with anchoring disabled ($q = 0$; Table 2). On FTRL, trace-conditioning alone yields a modest improvement in Solve-F1 from 31.60 to 31.85, with higher Solve-P but lower Solve-R. Adding component-aware SFT to prompt-only RL instead reduces Solve-F1 from 31.60 to 30.18 and Solve-P from 30.60 to 28.46. In contrast, adding component-aware SFT to trace-conditioned RL increases Solve-F1 from 31.85 to 33.21, with improvements in both Solve-R and Solve-P. Holding the SFT objective fixed, replacing prompt-only RL with trace-conditioned RL improves all three FTRL metrics, including a 3.03-point gain in Solve-F1. On BFCL, adding component-aware SFT improves the average score by 0.55 points with prompt-only RL and by 1.37 points with trace-conditioned RL, with the joint objective achieving the highest average of 20.44. These comparisons support the empirical complementarity of the two training signals across benchmarks: direct expert supervision is more beneficial when combined with trace-conditioned RL.

To examine whether trace conditioning remains beneficial in the main configuration ($q = 0.4$), we compare PACT with a variant that replaces trace-conditioned RL with prompt-only RL while retaining the same component-aware SFT and anchoring. Compared with this variant, PACT improves FTRL Solve-F1 by 4.73 and BFCL Avg by 4.81. These gains support the value of using expert traces as privileged RL context beyond their role as SFT supervision in the main training configuration. To further assess the benefit of task-matched expert context within joint SFT–RL training, we conduct a controlled single-iteration comparison of different RL conditioning contexts. Details of both the full-training comparison and this controlled experiment are provided in Appendix F.4.

Table 3: Ablation of privileged RL context using Qwen3-4B on FTRL and BFCL ($q = 0$). RE, TC, and TR denote reasoning, tool-calls, and tool responses. ✗ marks the removed component; SFT remains unchanged. BFCL Avg averages Multi-Turn, Search, and Memory accuracy.

			FTRL			BFCL
RE	**TC**	**TR**	Solve-R	Solve-P	Solve-F1	Avg
✗			35.31	27.46	26.12	17.89
	✗		39.19	28.52	31.11	16.31
		✗	31.32	25.27	25.94	14.93
	All		**40.45**	**32.94**	**33.21**	**20.44**

5.4 The Role of Privileged Trace Components as RL Context

Beyond the choice of training objectives, we examine which components of the expert trace provide useful context for RL policy optimization. We separately remove reasoning, tool-calls, and tool responses from the context used to compute the RL policy ratios, while keeping the SFT inputs, supervision masks, and loss coefficients unchanged (Table 3). Removing reasoning or tool

responses reduces Solve-F1 from 33.21 to 26.12 and 25.94, respectively, highlighting the value of both expert reasoning and environmental feedback as context for policy optimization. Removing tool-call context also lowers Solve-F1 to 31.11, indicating that expert action information remains useful even when tool-call SFT supervision is retained. Retaining all three components achieves the highest scores across all three FTRL metrics, supporting the use of complete expert traces as privileged context for RL updates. Detailed results are reported in Appendix F.5.

5.5 Choice of SFT Schedule

We next examine how to schedule direct SFT supervision for reasoning and tool-calls, while keeping the trace-conditioned RL setup unchanged (Table 4). With tool-call loss-scale annealing fixed, segment-level annealing for reasoning achieves a Solve-F1 of 33.21, compared with 30.60 for fixed supervision and 29.86 for token-level annealing. This suggests that gradually shortening the supervised reasoning prefixes at semantic boundaries is more effective than the compared alternatives. For tool-calls, with segment-level reasoning annealing fixed, loss-scale annealing improves Solve-F1 by 0.39 points over a fixed weight and by 4.38 points over token-level annealing. This pattern supports supervising complete tool-call spans throughout training while gradually relaxing the imitation constraint. The combination achieves the highest scores across all three FTRL metrics among the compared schedules, supporting different annealing strategies for reasoning and tool-calls. We provide additional schedule comparisons in Appendix F.6.

Table 4: Comparison of SFT schedules for reasoning and tool-calls using Qwen3-4B on FTRL with $q = 0$. "Fix" denotes no annealing for the corresponding component, "Seg A" anneals reasoning supervision by semantic segment, "Token A" anneals supervision by token count, and "Scale A" anneals the tool-call loss weight.

Reasoning	Tool-Call	Solve-R	Solve-P	Solve-F1
Fix	Scale A	37.90	28.40	30.60
Seg A	Fix	39.32	31.25	32.82
Token A	Scale A	34.77	28.08	29.86
Seg A	Token A	35.80	27.45	28.83
Seg A	**Scale A**	**40.45**	**32.94**	**33.21**

5.6 Effect of Prompt-only Anchoring

We vary the prompt-only anchoring probability q to examine the balance between trace co-training and prompt-only RL (Figure 4). Without prompt-only anchoring, optimization relies entirely on trace co-training, which may encourage reliance on the training-only trace context. Introducing a moderate proportion of prompt-only RL improves ToolHop performance, suggesting that direct optimization under the test-time input condition helps the policy generalize to prompt-only use. However, when q is large, fewer prompts receive trace-conditioned RL and component-aware SFT, and FTRL performance drops.

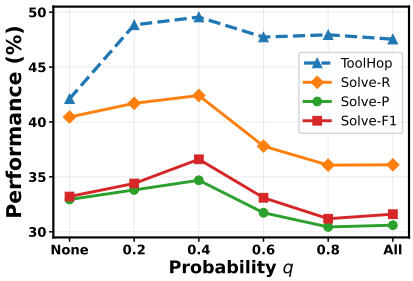

Figure 4: Effect of the prompt-only anchoring probability q on Qwen3-4B. Curves show FTRL Solve-R, Solve-P, and Solve-F1, and ToolHop AC. "None" and "All" denote $q = 0$ and $q = 1$, respectively.

6 Conclusion

In this work, we introduced PACT, a privileged trace co-training framework for multi-turn tool-use agents. PACT keeps rollout generation prompt-only and uses expert traces during optimization through trace-conditioned RL and component-aware SFT. Prompt-only anchoring adds a direct update signal under the inference-time input condition. Experiments on FTRL, BFCL, and ToolHop show improvements over the SFT- and RL-based baselines, highlighting the value of expert traces for optimization without using them as rollout hints.

AI USE STATEMENT

We used generative AI tools for grammar correction and language polishing of the paper. All AI-assisted manuscript edits were reviewed by the authors. We also used LLMs for expert-trace generation and for trace segmentation. LLMs are also used as the base models for training.

REPRODUCIBILITY STATEMENT

We provide the training data and implementation code in the supplementary material. Algorithm 1 describes the complete training procedure of PACT. Appendix A reports the training hyperparameters, baseline configurations, and evaluation settings. Appendix B specifies the reward computation and advantage estimation used for policy optimization. Appendix C details expert-trace construction, segmentation, supervision-prefix definitions, and prompt templates.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

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

Liang Chen, Xueting Han, Li Shen, Jing Bai, and Kam-Fai Wong. Beyond two-stage training: Cooperative sft and rl for llm reasoning. *arXiv preprint arXiv:2509.06948*, 2025a.

Zhixun Chen, Ming Li, Yuxuan Huang, Yali Du, Meng Fang, and Tianyi Zhou. Atlas: Agent tuning via learning critical steps. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 25334–25349, 2025b.

Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Ep0TtjVoap.

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

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116, 2023.

Minpeng Liao, Chengxi Li, Wei Luo, Wu Jing, and Kai Fan. Mario: Math reasoning with code interpreter output-a reproducible pipeline. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 905–924, 2024.

Mingyang Liu, Gabriele Farina, and Asuman Ozdaglar. UFT: Unifying supervised and reinforcement fine-tuning. *arXiv preprint arXiv:2505.16984*, 2025. URL https://arxiv.org/abs/2505.16984.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K Qiu, and Yuqing Yang. Agent lightning: Train any ai agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Lei Pang, Jun Luo, and Ruinan Jin. Tic-grpo: Provable and efficient optimization for reinforcement learning from human feedback. *arXiv preprint arXiv:2508.02833*, 2025.

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

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

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. *arXiv preprint arXiv:2602.04942*, 2026.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, et al. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *Advances in Neural Information Processing Systems*, 38, 2026.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *Advances in Neural Information Processing Systems*, 38:105523–105553, 2026.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Changle Qu, Sunhao Dai, Hengyi Cai, Jun Xu, Shuaiqiang Wang, and Dawei Yin. Matchtir: Fine-grained supervision for tool-integrated reasoning via bipartite matching. *arXiv preprint arXiv:2601.10712*, 2026. URL https://arxiv.org/abs/2601.10712.

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

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

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

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Agentbank: Towards generalized llm agents via fine-tuning on 50000+ interaction trajectories. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 2124–2141, 2024.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

Zihan Wang, Chi Gui, Xing Jin, Qineng Wang, Licheng Liu, Kangrui Wang, Shiqi Chen, Linjie Li, Zhengyuan Yang, Pingyue Zhang, et al. Ragen-2: Reasoning collapse in agentic rl. *arXiv preprint arXiv:2604.06268*, 2026.

Quan Wei, Siliang Zeng, Chenliang Li, William Brown, Oana Frunza, Wei Deng, Yuriy Nevmyvaka, Yang Katie Zhao, Alfredo Garcia, and Mingyi Hong. Reinforcing multi-turn reasoning in LLM agents via turn-level reward design and credit assignment. In *First Workshop on Multi-Turn Interactions in Large Language Models*, 2025. URL https://openreview.net/forum?id=drP7qVUnUt.

Minrui Xu, Zilin Wang, Mengyi DENG, Zhiwei Li, Zhicheng Yang, Xiao Zhu, Yinhong Liu, Boyu Zhu, Baiyu Huang, Chao Chen, Heyuan Deng, Fei Mi, Lifeng Shang, Xingshan Zeng, and Zhijiang Guo. Envfactory: Scaling tool-use agents via executable environments synthesis and robust rl. *arXiv preprint*, 2026. URL https://arxiv.org/abs/2605.18703.

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

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled rlvr. *arXiv preprint arXiv:2604.03128*, 2026.

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

Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, et al. Toolhop: A query-driven benchmark for evaluating large language models in multi-hop tool use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2995–3021, 2025.

Junjie Ye, Changhao Jiang, Zhengyin Du, Yufei Xu, Xuesong Yao, Zhiheng Xi, Xiaoran Fan, Qi Zhang, Tao Gui, Xuan-Jing Huang, et al. Feedback-driven tool-use improvements in large language models via automated build environments. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 2293–2323, 2026.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Xiangchi Yuan, Xiang Chen, Tong Yu, Dachuan Shi, Can Jin, Wenke Lee, and Saayan Mitra. Mitigating forgetting between supervised and reinforcement learning yields stronger reasoners. *arXiv preprint arXiv:2510.04454*, 2025.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Zhiwei Liu, Yihao Feng, et al. xlam: A family of large action models to empower ai agent systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11583–11597, 2025.

Wenhao Zhang, Yuexiang Xie, Yuchang Sun, Yanxi Chen, Guoyin Wang, Yaliang Li, Bolin Ding, and Jingren Zhou. On-policy RL meets off-policy experts: Harmonizing supervised fine-tuning and reinforcement learning via dynamic weighting. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=dCm9bBrk5d.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 414–431, 2025.

APPENDIX CONTENTS

A More Implementation Details **14**
A.1 Training Settings 14
A.2 Baseline Implementation Details 15
A.3 Evaluation Settings 15

B Training Reward and Advantage **16**

C Training Dataset Construction **17**
C.1 Data Sources and Access 17
C.2 Expert-Trace Construction 17
C.3 Reasoning-Prefix Definitions and Supervision 17
C.4 Prompt Templates 18

D Training Algorithm **20**

E A Conceptual Latent-Trace View of PACT **21**

F Additional Experiments and Methodological Details **22**
F.1 Extended Training Budget 22
F.2 Teacher Quality on a Common-Support Training Set 22
F.3 Offline Data Construction and Model Training Costs 22
F.4 Cross-Benchmark Objective Ablation 23
F.5 Cross-Benchmark Trace-Component Ablation 24
F.6 More Choices of SFT Schedule 25

G Limitations **26**

H Case Study **26**

A More Implementation Details

A.1 Training Settings

Our implementation is based on the VeRL framework (Sheng et al., 2024). We use GRPO-style policy optimization with the rewards and advantages described in Appendix B. For each training prompt, we sample $K = 8$ prompt-only rollouts at temperature 1.0, with a maximum of 10 interaction turns per rollout. Policy-model training and evaluation use four NVIDIA H200 141GB GPUs. The table below summarizes the default configuration for the main PACT experiments.

Table 5: Default training configuration for the main PACT experiments. The temperature and rollout count refer to training-time sampling.

Hyperparameter	Value
Data and Token Budgets	
Training batch size	256
Validation batch size	256
Maximum prompt length	7000
Maximum response length	23000
Maximum token length per GPU	30000
Optimization	
Learning rate	1×10^{-6}
PPO mini-batch size	32
KL loss coefficient β_{KL}	0.001
SFT loss scale β_{SFT}	0.1
Reasoning loss scale	1.0
Tool-call start scale $\lambda_a^{\mathrm{start}}$	0.8
Tool-call target end scale λ_a^{end}	0.2
Prompt-only anchoring probability q	0.4
Training Rollouts	
Rollout framework	vLLM
Sampling temperature	1.0
Rollouts per prompt K	8
Maximum interaction turns	10
GPU memory utilization	0.7
Default Training Budget	
Training updates	24

We fix the overall SFT coefficient at $\beta_{\mathrm{SFT}} = 0.1$ throughout training for PACT and the objective-ablation variants that include component-aware SFT. The reasoning and tool-call losses are normalized separately. The reasoning loss scale remains 1.0, while the supervised prefix within each assistant turn is annealed from L_4 to L_0 over five equal intervals of training progress. At L_0, reasoning tokens receive no direct SFT loss but remain in the context. Complete tool-call spans remain supervised, with their loss weight linearly annealed from 0.8 toward a target of 0.2. Prefix-level definitions are provided in Appendix C.3; additional SFT schedule comparisons are provided in Appendix F.6.

For the main experiments, $q = 0.4$ was selected based on training-time performance. We compare $q \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$ in Figure 4. The objective, privileged-context, and SFT-schedule ablations use $q = 0$ to disable prompt-only anchoring. The extended-training experiment in Appendix F.1 increases the training budget from 24 to 48 updates.

Privileged-context construction. We first format the task prompt using the model's chat template. Expert reasoning, tool calls, and formatted tool responses are tokenized separately, and their token IDs are concatenated after the formatted prompt without reapplying the chat template or adding an instruction introducing the expert trace. For likelihood computation, the original prompt-only rollout is appended after this privileged context. The expert trace is supplied as a single prefix, rather than interleaved with individual rollout turns. The literal strings user and assistant in the expert tool-response formatting are ordinary text, not chat-template role delimiters.

A.2 Baseline Implementation Details

All baselines in Table 1 start from the same Qwen3-1.7B or Qwen3-4B checkpoints. Table 6 gives the actual training budgets and the key hyperparameters of each baseline. For the non-OPSD on-policy baselines, the common rollout temperature is 1.0, the PPO mini-batch size is 32, the reference-policy KL coefficient is 0.001, and the entropy coefficient is 0.001.

Table 6: Baseline training parameters for the main comparison. B is the training batch size, K is the number of rollouts per prompt, and η is the policy learning rate. Updates refer to optimizer steps, not rollouts.

Baseline	**Updates**	**Key training parameters**
Vanilla	0	Unmodified Qwen3 checkpoint.
SFT	24	$B = 256$, $\eta = 10^{-6}$; full-trace cross-entropy.
GRPO	24	$B = 256$, $K = 8$, $\eta = 10^{-6}$; final-answer exact-match reward.
FTRL	24	$B = 256$, $K = 8$, $\eta = 10^{-6}$; Solve-F1 reward.
ToolRL	24	$B = 256$, $K = 8$, $\eta = 10^{-6}$; tool-format and call-correctness reward.
CHORD	24	$B = 256$, $K = 8$, $\eta = 10^{-6}$; fixed $\mu = 0.1$ and token weight $\phi(p) = p(1-p)$ on expert SFT.
OPSD	24	$B = 256$, $K = 8$, $\eta = 10^{-6}$, reverse-KL weight $\beta = 0.1$.
MatchTIR	24	$B = 256$, $K = 8$, $\eta = 10^{-6}$; tool-call matching and dual-level advantages ($\gamma = 0.9$).
SFT→MatchTIR	$24 + 24$	Full-trace SFT then MatchTIR RL; $B = 256$, $\eta = 10^{-6}$, and $K = 8$ in the RL stage.

A.3 Evaluation Settings

We evaluate on the released FTRL test split (Ye et al., 2026), the Multi-Turn, Search, and Memory subtasks of BFCL (Patil et al., 2025), and ToolHop (Ye et al., 2025). At evaluation time, the policy receives the task prompt and its ordinary interaction history, without privileged expert traces. Data-split separation and baseline access to expert traces are described in Appendix C.1.

Evaluation protocol. We perform one evaluation run per test instance on each benchmark. FTRL and ToolHop use greedy decoding (temperature 0) with thinking enabled and at most 10 model responses per instance; a response may contain multiple tool-calls. ToolHop is evaluated in the Free setting with a per-generation output limit of 4,096 tokens. BFCL uses a per-generation output cap of 4,096 tokens and follows the benchmark's native interaction and step limits.

FTRL. For test instance i, let n_i be the number of sub-questions, p_i the number of tool invocations, and c_i the number of successfully solved sub-questions. The per-instance solve precision P_i, solve rate R_i, and Solve-F1 F_i are

$$P_i = \begin{cases} c_i/p_i, & p_i > 0, \\ 1, & p_i = 0, \end{cases} \qquad R_i = \frac{c_i}{n_i}, \tag{14}$$

$$F_i = \begin{cases} \dfrac{2P_iR_i}{P_i + R_i}, & P_i + R_i > 0, \\ 0, & P_i + R_i = 0. \end{cases} \tag{15}$$

Solve-P measures tool-use precision, Solve-R measures task completeness, and Solve-F1 combines the two.

BFCL. We use the benchmark's accuracy-based evaluation for Multi-Turn, Search, and Memory. Unless otherwise specified, BFCL Avg is the arithmetic mean of these three subtask scores.

ToolHop. We report Answer Correctness (AC). For a test instance with reference answer a_i and final response o_i, the per-instance score is 1 if o_i contains a_i, and 0 otherwise:

$$\mathrm{AC}_i = \mathbf{1}[\mathrm{contains}(o_i, a_i)]\,. \tag{16}$$

Overall average. Unless otherwise specified, all reported scores are percentages. The final Avg in Table 1 gives equal weight to the three benchmarks:

$$\mathrm{Avg} = \frac{\mathrm{FTRL\ Solve\text{-}F1} + \mathrm{BFCL\ Avg} + \mathrm{ToolHop\ AC}}{3}. \tag{17}$$

B Training Reward and Advantage

We build on the matching-based rewards and dual-level advantage estimation of MatchTIR (Qu et al., 2026), with the implementation details specified below. Both prompt-only RL and trace-conditioned RL use the same reward targets and advantage estimates. For clarity, we fix a prompt x_i and omit its index i below.

Hard credit assignment. For each rollout, let $\mathcal{P} = \{p_1, \ldots, p_m\}$ denote its predicted tool-calls. The reference tool-call list $\mathcal{G} = \{g_1, \ldots, g_n\}$ contains the fixed ground-truth calls associated with x_i and supplies the matching targets for reward computation. These targets are shared across the teacher variants in Appendix F.2 and are not replaced when the teacher-generated trace h_i^* changes. For each pair (p_a, g_b), we compute a similarity score S_{ab} based on tool-name match, argument-key overlap, and exact argument-value match. Our implementation uses greedy one-to-one matching: candidate pairs are sorted by decreasing similarity, and a pair is accepted only if its score is positive and neither tool-call has already been matched. Let $X_{ab} = 1$ for an accepted pair and 0 otherwise. The reward for p_a is

$$r(p_a) = \begin{cases} S_{ab}, & \exists b,\ X_{ab} = 1, \\ -\lambda_{\mathrm{pen}}, & \text{otherwise}, \end{cases} \qquad \lambda_{\mathrm{pen}} = 0. \tag{18}$$

Turn-level and outcome rewards. For a tool-use turn t, we average the rewards of the predicted tool-calls in that turn:

$$r_t^{\mathrm{tool}} = \frac{1}{|\mathcal{P}_t|} \sum_{p \in \mathcal{P}_t} r(p), \tag{19}$$

with $r_t^{\mathrm{tool}} = 0$ when $\mathcal{P}_t$ is empty. For the final-answer turn, we use $r^{\mathrm{ans}} = \mathrm{F1}(\hat{y}, y)$ between the predicted answer $\hat{y}$ and reference answer y. The per-turn reward r_t is the tool-call reward for tool-use turns and the answer reward for the final turn.

Advantage estimation. Given K rollouts for the same prompt, let T_k be the number of turns in rollout k. We compute its trajectory-level reward and normalize it within the rollout group:

$$R_k = \sum_{t=1}^{T_k} r_{k,t}, \qquad A_k^g = \frac{R_k - \mathrm{mean}_{k'=1,\ldots,K}(R_{k'})}{\mathrm{std}_{k'=1,\ldots,K}(R_{k'}) + \epsilon_A}. \tag{20}$$

For each turn, the discounted future reward is

$$R_{k,t}^{\ell} = \sum_{u=t}^{T_k} \gamma^{u-t} r_{k,u}, \qquad \gamma = 0.9, \quad \epsilon_A = 10^{-6}. \tag{21}$$

Turn-level normalization uses only rollouts that reach turn t, indexed by $\mathcal{K}_t = \{k : T_k \geq t\}$:

$$A_{k,t}^{\ell} = \frac{R_{k,t}^{\ell} - \mu_t}{\sigma_t + \epsilon_A}, \qquad k \in \mathcal{K}_t. \tag{22}$$

When $|\mathcal{K}_t| > 1$, μ_t and σ_t are the mean and sample standard deviation of $\{R_{k',t}^{\ell} : k' \in \mathcal{K}_t\}$. Terminated rollouts are excluded from these statistics rather than zero-padded. When $|\mathcal{K}_t| = 1$, we set $\mu_t = 0$ and $\sigma_t = 1$; thus, the remaining rollout receives $A_{k,t}^{\ell} = R_{k,t}^{\ell}/(1 + \epsilon_A)$. For any model-generated token n in turn t, the final advantage averages the two components:

$$A_{k,n} = \frac{A_k^g + A_{k,t}^{\ell}}{2}. \tag{23}$$

Tool-response tokens are generated by the environment and are excluded from the policy loss.

C TRAINING DATASET CONSTRUCTION

We construct the training dataset from executable multi-turn tool-use tasks. Each raw example contains a user question, a list of available tool schemas, the corresponding executable tool implementations, and a reference answer. Since PACT requires process-level guidance rather than only final-answer labels, we first generate expert tool-use trajectories with a teacher model, and then convert these trajectories into segmented privileged traces for masked process supervision.

C.1 DATA SOURCES AND ACCESS

Expert-trace construction and model training use only released FTRL training prompts. FTRL evaluation uses its released test split; BFCL and ToolHop are used only for evaluation. Among the trained baselines, GRPO, FTRL, ToolRL, and MatchTIR do not use full expert traces as policy-conditioning context or direct SFT targets. SFT, CHORD, OPSD, and SFT→MatchTIR use the same verified expert-trace pool as PACT. This distinction concerns full-trace conditioning and direct supervision; the fixed reference tool-calls used as reward targets are described in Appendix B.

C.2 EXPERT-TRACE CONSTRUCTION

Expert trajectory generation. For each raw question, we prompt a teacher model to solve the task under the same tool-call interface used during training and inference. The teacher is instructed to obtain all information through tool-calls and to output the final answer only inside the `<answer></answer>` tag. After each generated tool-call, we execute the corresponding local tool implementation and append the tool observation back to the context. This iterative process continues until the teacher emits a final answer or reaches the maximum turn budget. We retain a trajectory only when the final answer, or the last tool observation, verifies against the reference answer.

We use the Qwen3-235B teacher generation as the main source of expert traces. We then remove invalid trajectories, including examples with invalid tool names, malformed parsed turns, tool responses containing role text, or tool responses that incorrectly contain tool-call tags. For training prompts without a verified Qwen3-235B trace, we additionally use xLAM-32B (Zhang et al., 2025) to improve trace coverage. Traces from both sources undergo the same execution-based verification and validity filtering before being included in the training set. The final training set contains 2,099 samples.

Hybrid segmentation of expert traces. To support component-aware SFT, we segment expert reasoning while retaining complete tool-call spans and tool observations in the trace. For each assistant turn, we extract the reasoning inside the `<think>` block, normalize role noise and whitespace, split the reasoning into paragraphs, and associate the turn with its structured tool-call and tool observation.

We apply a hybrid segmentation pipeline. Structurally simple turns are segmented with deterministic rules, while harder turns are sent to an LLM annotator. If both routes fail, we use a conservative fallback segmenter. Each segmented turn is annotated with semantic chunks such as `task_restate`, `tool_scan`, `tool_selection`, `required_arg_grounding`, `failure_reflection`, and `decision_summary`. The segmenter also detects explicit tool-call leakage, including literal JSON previews or phrases that introduce the exact action. Such spans are marked as `explicit_call_preview` and are excluded from eligible reasoning-supervision prefixes.

C.3 REASONING-PREFIX DEFINITIONS AND SUPERVISION

Eligible reasoning chunks. For each turn, we split the reasoning into ordered semantic chunks and retain the longest initial sequence of safe chunks ending immediately before the first explicit tool-call preview or other unsafe leakage span. Let this eligible sequence be $(C_1, \ldots, C_M)$. The

levels are cumulative left prefixes, not five fixed semantic categories:

$$\begin{aligned} L_0 &= \varnothing, \\ L_1 &= (C_1, \ldots, C_{\min(1,M)}), \\ L_2 &= (C_1, \ldots, C_{\min(2,M)}), \\ L_3 &= (C_1, \ldots, C_{\min(3,M)}), \\ L_4 &= (C_1, \ldots, C_M). \end{aligned} \tag{24}$$

All tuples are empty when $M = 0$. For $M \leq 3$, levels exceeding the available number of chunks repeat the longest eligible prefix. For $M > 3$, L_4 includes all eligible chunks, rather than only the first four. The training schedule divides progress into five equal intervals and uses $L_4 \to L_3 \to L_2 \to L_1 \to L_0$.

Worked example and loss masking. Consider a span segmented into task restatement (C_1), tool selection (C_2), argument grounding (C_3), a pre-call decision (C_4), and a literal tool-call preview (C_5). In this example, the first four chunks are eligible and the preview is excluded from the eligible reasoning-supervision prefix, yielding $L_0 = []$, $L_1 = [C_1]$, $L_2 = [C_1, C_2]$, $L_3 = [C_1, C_2, C_3]$, and $L_4 = [C_1, C_2, C_3, C_4]$.

The selected prefix specifies which reasoning tokens receive direct SFT supervision in Eq. 9. The loss of tool-call tokens is normalized separately from the reasoning loss and then scaled by the scheduled weight λ_a. L_0 means no direct reasoning-token SFT loss. Rollout generation remains prompt-only as described in Section 4.1.

C.4 Prompt Templates

Teacher prompt template. The prompt template used for expert trajectory generation is shown below. The available tool schemas are rendered inside the `<tools></tools>` block, and the teacher must serialize each function call inside `<tool_call></tool_call>` (Qu et al., 2026).

Prompt Template for Expert Trajectory Generation

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>`
{Tool List}
`</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>`
User Prompt:
Please call given tools to answer the question. Please note that all your information must be obtained by calling tools and not by answering the question directly.
If the call fails, you need to try to correct it and continue until you arrive at an answer.
Only output the final answer inside the `<answer></answer>` tag, without any explanations or extra information.
Question: {question}
Output:

LLM segmentation prompt. For difficult reasoning turns, we use an LLM annotator [1] to produce semantic segments for reasoning-prefix supervision. The annotator is not asked to answer the question; it only segments an existing teacher turn.

[1] We use ChatGPT-5.4 here.

Prompt Template for Trace Segmentation

You are a data annotation engine for segmenting tool-use reasoning turns into semantic chunks for autoregressive prefix-hint training.
Your job is NOT to answer the user question. Your job is to analyze one assistant turn and split its reasoning into contiguous semantic chunks that can be used as left-to-right prefix hints.
Important training setting:
- The downstream model is autoregressive.
- Only a LEFT PREFIX of the reasoning can be exposed as a hint.
- Therefore, segmentation must support prefix-level reveal.
- Chunks must preserve the original order.
- Chunks must be contiguous and non-overlapping.
Main goal:
Given the current turn's reasoning text, identify semantic stages such as task understanding, tool scanning or comparison, tool selection, required argument grounding, optional argument analysis, reflection after tool failure, fallback planning, final decision summary before action, and explicit tool-call preview.
Critical rule about leakage:
If the reasoning contains an explicit preview of the exact tool call, JSON, function name with all arguments, or a phrase that clearly introduces the literal action, mark that span as `explicit_call_preview`. This span should normally NOT be included in recommended prefix hint levels.
Return VALID JSON only. Do not include markdown. Do not paraphrase the original reasoning text. Do not invent missing paragraphs. Use paragraph ids to define chunks. If an intra-paragraph split is necessary, use `split_paragraph_id` and an exact `cut_before_text` anchor. Produce prefix levels `L0`, `L1`, `L2`, `L3`, and `L4`, where `L0` reveals no hint and `L4` is the strongest safe prefix before leakage.
Allowed chunk types:
`task_restate`, `tool_scan`, `tool_selection`, `required_arg_grounding`, `optional_arg_analysis`, `irrelevant_tool_rejection`, `failure_reflection`, `fallback_plan`, `decision_summary`, `explicit_call_preview`, `final_answer_grounding`, `other`.
Allowed turn patterns:
`initial_attempt`, `compare_then_call`, `retry_after_failed_call`, `post_observation_followup`, `final_answer_turn`, `other`.
Return JSON following this schema:
{ "turn_pattern": "...", "cleaning_actions": [{"type": "...", "detail": "..."}], "chunks": [{ "chunk_id": "C1", "chunk_type": "...", "paragraph_ids": [0, 1], "split_paragraph_id": null, "cut_before_text": null, "summary": "...", "keep_for_prefix": true, "hint_strength": "weak—medium—strong" }], "dangerous_spans": [{ "type": "explicit_call_preview", "paragraph_id": 4, "cut_before_text": "So, the tool call would be:" }], "recommended_prefix_levels": [{"level": "L0", "visible_chunks": []}, {"level": "L1", "visible_chunks": ["C1"]}, {"level": "L2", "visible_chunks": ["C1", "C2"]}, {"level": "L3", "visible_chunks": ["C1", "C2", "C3"]}, {"level": "L4", "visible_chunks": ["C1", "C2", "C3", "C4"]}], "quality_flags": ["clean_structure"] }

D TRAINING ALGORITHM

Algorithm 1 gives the training procedure of PACT. Rollouts are generated without expert-trace conditioning; rewards and advantages are computed as described in Appendix B. Each prompt selects either trace co-training or prompt-only RL, with one gate shared by its K rollouts. Component-aware SFT follows the masks and annealing rules in Section 4.3, while retaining the full teacher-forced context. The per-prompt SFT contributions are scaled so that their batch average recovers the separately normalized loss in Eq. 10 over trace-branch prompts; a component with no selected tokens contributes zero.

Algorithm 1 PACT Training

Require: Policy π_θ, reference policy π_{ref}, training data $\{(x_i, h_i^*)\}$, rollout count K, anchoring probability q, SFT weight β_{SFT}.

for each training update s **do**
  Freeze $\pi_{\mathrm{old}} \leftarrow \pi_\theta$ and sample a batch $\mathcal{B}$.
  Sample $\tau_{i,k} \sim \pi_{\mathrm{old}}(\cdot \mid x_i)$ for each $i \in \mathcal{B}$ and $k = 1, \ldots, K$.
  Compute rollout rewards and token-level advantages $A_{i,k,n}$ (Appendix B).
  Sample one gate $z_i \sim \mathrm{Bernoulli}(q)$ for each prompt.
  Update reasoning-prefix masks and tool-call weight $\lambda_a(s)$ (Section 4.3).
  Compute SFT contributions $\mathcal{L}_{\mathrm{SFT}}(i; s)$ for $z_i = 0$, using separate component normalization (Eq. 10).
  for each prompt $i \in \mathcal{B}$ **do**
    if $z_i = 0$ **then**
      Set the privileged context $c_i^h = [x_i; h_i^*]$.
      $r_{i,k,n}^h \leftarrow \dfrac{\pi_\theta(\tau_{i,k,n} \mid c_i^h, \tau_{i,k,<n})}{\pi_{\mathrm{old}}(\tau_{i,k,n} \mid c_i^h, \tau_{i,k,<n})}, \quad n \in \mathcal{M}_{i,k}.$
      Compute $\mathcal{L}_{\mathrm{RL}}^h(i)$ from $r_{i,k,n}^h$ and $A_{i,k,n}$, with clipping and KL regularization (Eq. 8).
      $\mathcal{L}_i^{\mathrm{trace}} \leftarrow \mathcal{L}_{\mathrm{RL}}^h(i) + \beta_{\mathrm{SFT}} \mathcal{L}_{\mathrm{SFT}}(i; s)$.
      $\mathcal{L}_i \leftarrow \mathcal{L}_i^{\mathrm{trace}}$.
    else
      Compute $\mathcal{L}_{\mathrm{RL}}^x(i)$ using prompt-only context and the same rollout advantages.
      $\mathcal{L}_i \leftarrow \mathcal{L}_{\mathrm{RL}}^x(i)$ ▷ No SFT loss
    end if
  end for
  Minimize $\mathcal{L} = \dfrac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathcal{L}_i$ with an optimizer step on θ.
end for

E A CONCEPTUAL LATENT-TRACE VIEW OF PACT

We define an auxiliary latent-trace model to provide a conceptual interpretation of the two trace-based training signals in PACT: learning from expert traces and using them as context for updating self-generated rollouts.

Auxiliary latent-trace model. For a fixed prompt x, let $p_\theta(h \mid x)$ be a normalized distribution over discrete process traces. Let $P_\theta(\tau \mid x, h)$ denote the trajectory distribution induced by π_θ conditioned on $[x; h]$ and an environment whose transition probabilities do not depend on θ. We assume a finite interaction horizon and include the generation and termination rules in these distributions. We define the auxiliary mixture distribution

$$\widetilde{P}_\theta(\tau \mid x) := \sum_h p_\theta(h \mid x) P_\theta(\tau \mid x, h). \tag{25}$$

The mixture $\widetilde{P}_\theta$ is introduced for this conceptual analysis; it is not assumed to equal the deployed prompt-only policy. PACT generates rollouts directly from the prompt-only policy, without first sampling an auxiliary trace.

An expert-trace lower bound. For an expert trace h^* with positive joint probability for the rollout under consideration, retaining its term in the marginal gives

$$\log \widetilde{P}_\theta(\tau \mid x) \geq \underbrace{\log p_\theta(h^* \mid x)}_{\text{trace likelihood}} + \underbrace{\log P_\theta(\tau \mid x, h^*)}_{\text{trace-conditioned rollout likelihood}}. \tag{26}$$

The two terms highlight complementary roles for an expert trace: learning the expert process itself and using that process as context for evaluating a sampled rollout. This provides a conceptual motivation for combining direct process supervision with trace-conditioned policy updates; the connections to the implemented objectives are detailed below.

Connection to component-aware SFT. The trace-likelihood term motivates learning the expert process itself. In PACT, component-aware SFT serves this role by supervising selected reasoning prefixes and complete tool-call spans (Eq. 10). Tool observations and unsupervised reasoning tokens remain in the teacher-forced context. Selective masks, separate component normalization, and an annealed tool-call weight define a structured supervision objective for learning selected components of the expert process.

Connection to trace-conditioned RL. The conditional likelihood term motivates using the expert process as context for evaluating self-generated behavior. For a fixed rollout, the parameter-dependent factors in $P_\theta(\tau \mid x, h^*)$ are the policy-generated token probabilities, including any policy-controlled termination decisions; environment-transition factors contribute no parameter gradient. PACT uses these token probabilities in its clipped policy ratios, retaining the advantages of the original prompt-only rollouts (Eq. 8).

Interpretation of joint training. This view motivates two related uses of the same expert trace: direct supervision of its model-generated components and privileged conditioning of reward-based updates on autonomous rollouts. PACT couples these signals through shared policy parameters, and Table 2 evaluates their empirical complementarity.

F Additional Experiments and Methodological Details

F.1 Extended Training Budget

We compare SFT, MatchTIR, and PACT on Qwen3-1.7B at 24 and 48 training updates, keeping the other settings fixed within each method. Table 7 reports the results for both budgets.

Table 7: Extended training on Qwen3-1.7B.

Updates	Method	FTRL Solve-F1	ToolHop AC	BFCL
24	SFT	17.66	23.32	8.16
24	MatchTIR	21.43	32.26	9.18
24	PACT	22.93	34.47	11.84
48	SFT	16.68	21.41	6.79
48	MatchTIR	23.42	38.09	8.38
48	PACT	27.17	40.50	12.62

At 48 updates, PACT remains ahead of MatchTIR by 3.75 on FTRL Solve-F1, 2.41 on ToolHop AC, and 4.24 on BFCL. Compared with 24 updates, PACT improves from 22.93 to 27.17 on FTRL, from 34.47 to 40.50 on ToolHop, and from 11.84 to 12.62 on BFCL. In contrast, extending SFT training reduces performance across all three benchmarks. These results show that PACT retains and generally strengthens its advantage over the compared baselines under a larger training budget.

F.2 Teacher Quality on a Common-Support Training Set

Since Qwen3-235B provides the primary source of expert traces in our main training set, we further study the effect of teacher quality in a controlled setting. We compare traces generated by Qwen3-8B and Qwen3-235B on a common-support subset. Their verified-trace acceptance rates are 55.4% and 73.0%, respectively. To separate trace quality from differences in accepted prompts, we form a common-support set of 1,096 training prompts for which both teachers produce a verified valid trace. All six runs in Table 8 use the same subset, a Qwen3-4B student, the same trace count, and matched training-budget and evaluation settings. Within each method, only the teacher trace source changes.

Table 8: Teacher-source comparison on the same training subset.

	FTRL Solve-F1		ToolHop AC	
Method	8B	235B	8B	235B
SFT	24.66	26.15	30.46	31.76
SFT→MatchTIR	27.72	28.88	36.21	37.39
PACT	**28.65**	**31.76**	**36.68**	**40.40**

The stronger teacher improves PACT by 3.11 points on FTRL and 3.72 points on ToolHop. With the 8B teacher, PACT still has the highest scores among the three controlled methods, exceeding SFT→MatchTIR by 0.93 and 0.47 points, respectively. Thus, PACT can be trained with a much smaller teacher and still retain advantages over the matched trace-using baselines.

F.3 Offline Data Construction and Model Training Costs

Offline construction. Expert-trace generation, execution-based verification, and segmentation are performed before model training. Table 9 reports the Qwen3-235B and Qwen3-8B construction costs. The usage costs are calculated based on OpenRouter's pricing.

Model training. All measured main Qwen3-4B training jobs use four H200 GPUs. GPU-hours equal the number of allocated GPUs multiplied by the complete wall time of the successful job. For SFT→MatchTIR, both stages are included. Wall times in Table 10 are displayed at minute resolution, whereas GPU-hours are computed from the full durations.

Table 9: Offline trace-construction API usage. Token counts are in millions; usage costs are in USD.

Stage	Input tokens	Output tokens	Usage cost
Qwen3-235B teacher			
Qwen3-235B trace generation	62.1	8.97	44.59
ChatGPT-5.4 segmentation-period calls	10.5	2.84	55.09
235B subtotal	**72.6**	**11.81**	**99.68**
Qwen3-8B teacher			
Qwen3-8B trace generation	44.20	5.72	7.77
ChatGPT-5.4 trace segmentation	4.84	1.38	31.60
8B subtotal	**49.04**	**7.10**	**39.37**

Table 10: Measured Qwen3-4B training cost on four H200 GPUs. Offline trace construction is excluded and reported separately above.

Method	Wall time (h:min)	H200 GPU-hours
MatchTIR	14:31	58.11
SFT→MatchTIR	01:23 + 11:46	52.68
PACT	16:16	65.13

In these runs, PACT uses 12.1% more GPU-hours than MatchTIR and 23.6% more than SFT→MatchTIR. At deployment, the model receives the task prompt and its ordinary interaction history: it does not require a teacher, privileged expert traces, or additional trace-generation or segmentation calls.

F.4 Cross-Benchmark Objective Ablation

We evaluate the four Qwen3-4B objective variants from the main ablation on BFCL as well as FTRL. All four use $q = 0$, disabling prompt-only anchoring, so that the comparison isolates the trace-conditioned RL surrogate and component-aware SFT. The variants are A: $\mathcal{L}^{x}_{\mathrm{RL}}$; B: $\mathcal{L}^{h}_{\mathrm{RL}}$; C: $\mathcal{L}^{x}_{\mathrm{RL}} + \beta_{\mathrm{SFT}}\mathcal{L}_{\mathrm{SFT}}$; and D: $\mathcal{L}^{h}_{\mathrm{RL}} + \beta_{\mathrm{SFT}}\mathcal{L}_{\mathrm{SFT}}$.

Table 11: Objective ablation on Qwen3-4B with $q = 0$. BFCL Avg is the arithmetic mean of Multi-Turn, Search, and Memory accuracy.

	FTRL			BFCL			
Variant	Solve-R	Solve-P	Solve-F1	Multi-Turn	Search	Memory	Avg
A	36.10	30.60	31.60	24.38	**17.50**	14.62	18.83
B	35.29	31.42	31.85	25.37	17.00	14.84	19.07
C	36.36	28.46	30.18	26.38	16.50	15.27	19.38
D	**40.45**	**32.94**	**33.21**	**26.62**	**17.50**	**17.20**	**20.44**

Variant D achieves the highest scores on all three FTRL metrics and the highest BFCL average, with the highest Multi-Turn and Memory scores and a tie with A on Search. On FTRL, adding component-aware SFT to prompt-only RL changes Solve-F1 by -1.42 points, whereas adding it to trace-conditioned RL improves Solve-F1 by 1.36 points. On BFCL, the corresponding gains are 0.55 and 1.37 points. These results support the empirical benefit of coupling the two losses across benchmarks.

Comparison under the main configuration. We further compare variants C and D with prompt-only anchoring enabled ($q = 0.4$). Both use Qwen3-4B trained for 24 updates, with the same component-aware SFT and anchoring scheme. Under this setting, variant D corresponds to the main PACT configuration. Table 12 reports the results.

Variant D outperforms C on all three FTRL metrics and all three BFCL subtasks, supporting the benefit of privileged RL conditioning under the main training configuration.

Table 12: Comparison of variants C and D with prompt-only anchoring enabled ($q = 0.4$). Both variants use the same component-aware SFT. BFCL Avg averages Multi-Turn, Search, and Memory accuracy.

	FTRL			BFCL			
Variant	Solve-R	Solve-P	Solve-F1	Multi-Turn	Search	Memory	Avg
C	36.62	31.16	31.87	21.50	10.00	17.85	16.45
D (PACT)	**42.41**	**34.69**	**36.60**	**27.50**	**18.00**	**18.28**	**21.26**

Controlled updates with matched and mismatched trace contexts. To further examine the role of task-matched expert context within joint SFT–RL training, we conduct a controlled update experiment starting from the same Qwen3-4B checkpoint obtained after 12 training iterations of PACT with $q = 0.4$. We collect a fixed batch of 256 prompts with eight prompt-only rollouts per prompt. All conditions reuse these rollouts, rewards, and advantages.

We compare three RL conditioning settings: prompt-only context (P), the corresponding expert trace (M, matched), and an expert trace from another task (X, mismatched). Mismatched traces are selected to approximately match trace length and tool sets, although tool sets are not always identical. The intervention affects only the context used for RL likelihood evaluation; the correct expert SFT targets and reward references remain unchanged. Each condition recomputes both old- and current-policy likelihoods under its own context and performs one training iteration using a freshly initialized AdamW optimizer, with the same learning rate and loss coefficients.

We evaluate both $q = 0$ and $q = 0.4$. For $q = 0.4$, all three conditions share the same prompt-level anchoring assignment. Anchored prompts receive standard prompt-only RL without SFT; the remaining prompts use the corresponding P/M/X context with identical expert SFT inputs and supervision masks. For $q = 0$, no prompts are assigned to the anchoring branch. After this single training iteration, we evaluate each updated policy on FTRL without expert traces. Table 13 reports the results.

Table 13: Controlled RL-context comparison from a shared PACT checkpoint. All conditions use the same sampled rollouts, rewards, advantages, and correct expert SFT targets. The SFT activation rule follows the anchoring setting. Scores are percentages; bold indicates the best post-update score in each column within each q setting.

q	RL Context	Solve-R	Solve-P	Solve-F1
—	Before update	36.41	30.76	30.89
0	Prompt-only (P)	36.98	29.34	30.52
	Mismatched trace (X)	36.45	29.09	29.95
	Matched trace (M)	**37.21**	**31.39**	**31.22**
0.4	Prompt-only (P)	36.58	28.29	29.57
	Mismatched trace (X)	35.18	29.81	29.46
	Matched trace (M)	**38.76**	**31.17**	**31.94**

Matched context achieves the highest post-update scores on all three FTRL metrics under both anchoring settings. Its Solve-F1 exceeds prompt-only context by 0.70 and 2.37 points at $q = 0$ and $q = 0.4$, respectively, and mismatched context by 1.27 and 2.48 points. At this checkpoint, matched context also improves all three metrics over the pre-update model. While the full-training comparisons in Tables 11 and 12 evaluate the overall effects of different training objectives, this experiment compares RL contexts from a shared checkpoint with rollouts, rewards, advantages, and expert SFT supervision held fixed. The results provide local evidence that task-matched expert context during RL updates can improve subsequent prompt-only performance.

F.5 Cross-Benchmark Trace-Component Ablation

We examine the role of individual expert-trace components as privileged RL context using Qwen3-4B with prompt-only anchoring disabled ($q = 0$). Following the setup of Table 3, we separately

remove reasoning, tool-calls, or tool responses from the context used to compute the RL policy ratios. The SFT inputs, supervision masks, and loss coefficients remain unchanged across variants, so the comparison concerns the role of each component as RL context rather than as a direct SFT target. Table 14 presents the detailed results.

Table 14: Privileged trace-component ablation using Qwen3-4B on FTRL and BFCL ($q = 0$). RE, TC, and TR denote reasoning, tool-calls, and tool responses. ✗ marks the removed component; SFT remains unchanged. BFCL Avg is the arithmetic mean of Multi-Turn, Search, and Memory accuracy.

			FTRL			BFCL			
RE	TC	TR	Solve-R	Solve-P	Solve-F1	Multi-Turn	Search	Memory	Avg
✗			35.31	27.46	26.12	26.12	14.00	13.55	17.89
	✗		39.19	28.52	31.11	24.62	14.00	10.32	16.31
		✗	31.32	25.27	25.94	23.75	10.50	10.54	14.93
	All		**40.45**	**32.94**	**33.21**	**26.62**	**17.50**	**17.20**	**20.44**

On FTRL, removing reasoning or tool responses lowers Solve-F1 from 33.21 to 26.12 and 25.94, respectively, while removing tool-calls lowers it to 31.11. The larger drops from removing reasoning and tool responses highlight the value of expert reasoning and environmental feedback as context for policy optimization. The drop from removing tool-calls shows that expert action information remains useful when tool-call SFT supervision is retained. Using the complete trace achieves the highest scores on all three FTRL metrics among the compared variants.

F.6 More Choices of SFT Schedule

We compare SFT schedules with Qwen3-4B on FTRL, keeping trace-conditioned RL fixed and prompt-only anchoring disabled ($q = 0$). Table 4 compares schedules within each component, while Table 15 includes single-component supervision and joint token-count annealing.

Table 15: Additional comparison of SFT schedule designs using Qwen3-4B on FTRL. "Seg A" denotes annealing over semantic reasoning segments, "Scale A" denotes annealing the tool-call loss scale, and "Full Token A" denotes token-count annealing over concatenated reasoning and tool-call tokens.

Reasoning	**Tool-Call**	**Solve-R**	**Solve-P**	**Solve-F1**
Seg A	None	35.11	32.67	31.22
None	Scale A	35.01	25.51	27.69
Full Token A		38.76	29.65	31.91
Seg A	**Scale A**	**40.45**	**32.94**	**33.21**

For reasoning, *Fix* supervises the L_4 prefix retained by preprocessing in each assistant turn throughout training, with a loss scale of 1.0. For tool-calls, *Fix* supervises complete spans with $\lambda_a = 1.0$. *Seg A* and *Scale A* follow the semantic-prefix and tool-call weight schedules in Section 4.3. For the two *Token A* variants in Table 4, a target component of N tokens within an assistant turn retains direct supervision on its first

$$N_{\text{sup}}(s) = \left\lfloor N \left(1 - \frac{s-1}{24}\right) \right\rfloor, \qquad s = 1, \ldots, 24, \tag{27}$$

tokens. This mask applies to reasoning text in *Token A / Scale A*, and to the concatenated tool-call text within each turn in *Seg A / Token A*; the latter assigns selected tool-call tokens a fixed weight of 1.0. Unselected tokens remain in the input context. The retained fraction decreases from 1 to $1/24$ before rounding. *None* omits the corresponding SFT component. The outer SFT coefficient remains $\beta_{\text{SFT}} = 0.1$.

The full component-aware schedule achieves higher Solve-F1 than reasoning-only or tool-call-only supervision (33.21 versus 31.22 and 27.69). *Full Token A*, which anneals supervision over concatenated reasoning and tool-call tokens, improves Solve-F1 over reasoning-only supervision to 31.91,

while reducing Solve-P from 32.67 to 29.65. The component-aware schedule achieves the highest scores on all three metrics in Table 15, supporting separate schedules for reasoning and tool-calls.

G LIMITATIONS

This work has several limitations. First, PACT relies on expert tool-use traces during training. These traces are used only as privileged optimization signals and are not needed at inference time, but their construction, verification, and segmentation introduce additional offline data-preparation costs. Second, our current task focuses on agent–tool interactions rather than ongoing dialogue with users. Extending PACT to settings where agents seek clarification and adapt their tool use to user feedback is a promising direction for future work. Finally, PACT uses predefined schedules for reasoning-prefix supervision and tool-call loss weighting, together with a fixed prompt-only anchoring probability. Adapting these choices to training dynamics remains an avenue for future work.

H CASE STUDY

We provide a case study to compare the multi-turn tool-use behavior of the vanilla model, MatchTIR, and PACT. For readability, we truncate the reasoning text but keep the full tool-calls, tool responses, and final answers. Incorrect or less accurate parts are highlighted in red.

Table 16: Case study comparison on the same prompt. Scores are reported on a 0–1 scale.

Model	Solve-P	Solve-R	Solve-F1
Vanilla Qwen3-4B	0.29	0.33	0.31
MatchTIR	**1.00**	0.67	0.80
PACT	**1.00**	**1.00**	**1.00**

Case Study Prompt

Question.
What is the quotient of the area of the reserve located near the tribe whose ceremonial dance has been declared an intangible cultural heritage in hectares, and the birth year of the scientist who discovered oxygen?

Reference Answer (not provide).
The correct reserve is Serengeti National Park, whose area is 1,500,000 hectares. The scientist who discovered oxygen is Joseph Priestley, whose birth year is 1733. Therefore, the quotient is:

$$\frac{1{,}500{,}000}{1733} \approx 865.55.$$

Answer: $\boxed{865.55}$

Vanilla Qwen3-4B

Turn 1: Identify the tribe
Assistant Reasoning.

```
Okay, let's tackle this question step by step. The user is asking for
     the quotient of two values: the area of a reserve near a tribe
    whose ceremonial dance is an intangible cultural heritage, in
    hectares, divided by the birth year of the scientist who
    discovered oxygen. ...
```

Assistant Tool-Call.

```
{"name": "cultural_heritage_locator", "arguments": {"heritage_type":
    "dance", "include_unesco": true}}
```

Tool Response.

```
The Maasai tribe is known for their traditional dance, which is a
    significant cultural heritage.
```

Turn 2: Search for a nearby reserve
Assistant Reasoning.

```
Okay, so the user wants the quotient of the area of a reserve near
    the Maasai tribe's location divided by the birth year of the
    oxygen discoverer. ...
```

Assistant Tool-Call.

```
{"name": "nearby_reserve_locator", "arguments": {"cultural_location":
     "Maasai", "region": "Tanzania"}}
```

Tool Response.

```
No known reserves found near the specified location.
```

Turn 3: Retry with Kenya
Assistant Reasoning.

```
Okay, the user is asking for the quotient of the area of a reserve
    near the Maasai tribe's location and the birth year of the oxygen
     discoverer. But when I tried to find reserves near the Maasai in
     Tanzania, there were none. ...
```

Assistant Tool-Call.

```
{"name": "nearby_reserve_locator", "arguments": {"cultural_location":
     "Maasai", "region": "Kenya", "reserve_type": "cultural"}}
```

Tool Response.

```
No known reserves found near the specified location.
```

Turn 4: Retry with Maasai Mara
Assistant Reasoning.

```
Okay, the user is asking for the quotient of the area of a reserve
    near the Maasai tribe's location and the birth year of the oxygen
     discoverer. But both attempts to find reserves near the Maasai
    in Tanzania and Kenya didn't return any results. ...
```

Assistant Tool-Call.

```
{"name": "nearby_reserve_locator", "arguments": {"cultural_location":
     "Maasai Mara", "region": "Kenya", "reserve_type": "cultural"}}
```

Tool Response.

```
No known reserves found near the specified location.
```

Turn 5: Look up birth year
Assistant Reasoning.

```
Okay, the user is asking for the quotient of the area of a reserve
    near the Maasai tribe's location and the birth year of the oxygen
     discoverer. But all attempts to find reserves near the Maasai
    have failed. ...
```

Assistant Tool-Call.

{"name": "historical_fact_lookup", "arguments": {"person_name": "
Joseph Priestley", "fact_type": "birth_year"}}

Tool Response.

{"result": "1733"}

Turn 6: Use an incorrect reserve
Assistant Reasoning.

Okay, let's see. The user is asking for the quotient of the area of a reserve near the Maasai tribe's location divided by the birth year of the oxygen discoverer. ...

Assistant Tool-Call.

{"name": "area_calculator", "arguments": {"location_name": "Maasai
Mara National Reserve", "unit": "hectares"}}

Tool Response.

{"location_name": "Maasai Mara National Reserve", "area": 100000, "
unit": "hectares", "include_water_bodies": true, "precision_level
": 3, "data_source": "satellite"}

Turn 7: Final calculation
Assistant Reasoning.

Okay, let's see. The user wants the quotient of the area of a reserve near the Maasai tribe's location divided by the birth year of the oxygen discoverer. ...

Assistant Tool-Call.

{"name": "advanced_calculator", "arguments": {"operation": "division
", "operand1": 100000, "operand2": 1733}}

Tool Response.

{"result": "57.70"}

Assistant Reasoning.

Okay, let's wrap this up. The user asked for the quotient of the area of a reserve near the Maasai tribe's location (in hectares) divided by the birth year of the oxygen discoverer. ...

Assistant Final Answer.

The quotient of the area of the Maasai Mara National Reserve (100,000 hectares) divided by the birth year of Joseph Priestley (1733) is **57.70**.

Answer: 57.70

Error Summary.

The vanilla model is derailed by failed reserve-search attempts and then switches to the wrong reserve, Maasai Mara National Reserve. As a result, it uses the wrong area, 100,000 hectares, instead of the correct Serengeti National Park area of 1,500,000 hectares. The birth year and calculator call are correct, but the final quotient is wrong because the reserve and area are incorrect.

MatchTIR

Turn 1: Identify the tribe
Assistant Reasoning.

```
Okay, let's tackle this question step by step. The user is asking for
     the quotient of two values: the area of a specific reserve near
    a tribe with a ceremonial dance declared as intangible cultural
    heritage, in hectares, divided by the birth year of the scientist
     who discovered oxygen. ...
```

Assistant Tool-Call.

```
{"name": "cultural_heritage_locator", "arguments": {"heritage_type":
    "dance", "include_unesco": true}}
```

Tool Response.

```
The Maasai tribe is known for their traditional dance, which is a
    significant cultural heritage.
```

Turn 2: Locate the reserve
Assistant Reasoning.

```
Okay, so the user wants the quotient of the area of a reserve near
    the Maasai tribe's ceremonial dance (which is intangible cultural
     heritage) in hectares, divided by the birth year of the
    scientist who discovered oxygen. ...
```

Assistant Tool-Call.

```
{"name": "nearby_reserve_locator", "arguments": {"cultural_location":
     "Maasai tribe", "radius": 100, "region": "Kenya"}}
```

Tool Response.

```
Serengeti National Park
```

Turn 3: Get reserve area
Assistant Reasoning.

```
Okay, so the user wants the quotient of the area of the Serengeti
    National Park (near the Maasai tribe) in hectares divided by the
    birth year of the scientist who discovered oxygen. ...
```

Assistant Tool-Call.

```
{"name": "area_calculator", "arguments": {"location_name": "Serengeti
     National Park", "unit": "hectares"}}
```

Tool Response.

```
{"location_name": "Serengeti National Park", "area": 1500000, "unit":
     "hectares", "include_water_bodies": true, "precision_level": 3,
    "data_source": "satellite"}
```

Turn 4: Get birth year
Assistant Reasoning.

```
Okay, so the area of Serengeti National Park is 1,500,000 hectares.
    Now I need to find the birth year of the scientist who discovered
     oxygen. From what I remember, the discovery of oxygen is often
    attributed to Joseph Priestley, but I should verify this using
    the historical_fact_lookup tool. ...
Missing step:  the model does not call element_discovery_info to
    verify Joseph Priestley is the oxygen discoverer.
```

Assistant Tool-Call.

```
{"name": "historical_fact_lookup", "arguments": {"person_name": "
    Joseph Priestley", "fact_type": "birth_year"}}
```

Tool Response.

```
{"result": "1733"}
```

Turn 5: Final answer
Assistant Reasoning.

```
Okay, so the area of Serengeti National Park is 1,500,000 hectares,
    and the birth year of Joseph Priestley is 1733. The user wants
    the quotient of these two values.

Wait, the quotient is area divided by birth year. So that's 1,500,000
     / 1733. Let me calculate that. ...
Missing step:  the model does not call advanced_calculator for the
    final division and instead computes an approximate value
    manually.
```

Assistant Final Answer.

```
The quotient is calculated by dividing the area of Serengeti National
     Park (1,500,000 hectares) by the birth year of Joseph Priestley
    (1733).

$$
\frac{1,500,000}{1733} \approx 865.5
$$

**Final Answer:** 865.5
```

Error Summary.

```
Although MatchTIR follows the main correct path, it misses two
    expected tool-use steps:  verifying the oxygen discoverer with
    element_discovery_info and computing the final quotient with
    advanced_calculator.  Therefore, its answer is only approximate
    and its tool-use trajectory is incomplete.
```

PACT Output

Turn 1: Identify the tribe
Assistant Reasoning.

```
Okay, let's break down the user's question step by step. The user is
    asking for the quotient of two values: the area of a reserve near
     a tribe with a ceremonial dance declared intangible cultural
    heritage, and the birth year of the scientist who discovered
    oxygen. ...
```

Assistant Tool-Call.

```
{"name": "cultural_heritage_locator", "arguments": {"heritage_type":
    "dance", "include_unesco": true}}
```

Tool Response.

```
The Maasai tribe is known for their traditional dance, which is a
    significant cultural heritage.
```

Turn 2: Locate the reserve
Assistant Reasoning.

```
Okay, so the user wants the quotient of the area of a reserve near
    the Maasai tribe's ceremonial dance (which is intangible cultural
     heritage) in hectares, divided by the birth year of the
    scientist who discovered oxygen. ...
```

Assistant Tool-Call.

```
{"name": "nearby_reserve_locator", "arguments": {"cultural_location":
     "Maasai tribe", "radius": 100}}
```

Tool Response.

```
Serengeti National Park
```

Turn 3: Get reserve area
Assistant Reasoning.

```
Okay, so the user wants the quotient of the area of the Serengeti
    National Park (near the Maasai tribe) in hectares divided by the
    birth year of the scientist who discovered oxygen. ...
```

Assistant Tool-Call.

```
{"name": "area_calculator", "arguments": {"location_name": "Serengeti
     National Park", "unit": "hectares"}}
```

Tool Response.

```
{"location_name": "Serengeti National Park", "area": 1500000, "unit":
     "hectares", "include_water_bodies": true, "precision_level": 3,
    "data_source": "satellite"}
```

Turn 4: Identify oxygen discoverer
Assistant Reasoning.

```
Okay, so the user wants the quotient of the area of Serengeti
    National Park in hectares divided by the birth year of the
    scientist who discovered oxygen. First, I already have the area
    of Serengeti National Park as 1,500,000 hectares. ...
```

Assistant Tool-Call.

```
{"name": "element_discovery_info", "arguments": {"element_name": "
    oxygen", "include_year": true}}
```

Tool Response.

```
{"year": 1774, "discoverer": "Joseph Priestley", "element_name": "
    Oxygen"}
```

Turn 5: Get birth year
Assistant Reasoning.

```
Okay, so the user wants the quotient of the area of Serengeti
    National Park in hectares divided by the birth year of the
    scientist who discovered oxygen. First, I already have the area
    of Serengeti National Park as 1,500,000 hectares. ...
```

Assistant Tool-Call.

```
{"name": "historical_fact_lookup", "arguments": {"person_name": "
    Joseph Priestley", "fact_type": "birth_year"}}
```

Tool Response.

```
{"result": "1733"}
```

Turn 6: Final calculation
Assistant Reasoning.

```
Okay, let's see. The user wants the quotient of the area of Serengeti
     National Park in hectares divided by the birth year of the
    scientist who discovered oxygen. ...
```

Assistant Tool-Call.

```
{"name": "advanced_calculator", "arguments": {"operation": "division
    ", "operand1": 1500000, "operand2": 1733}}
```

Tool Response.

```
{"result": "865.55"}
```

Turn 7: Final answer
Assistant Reasoning.

```
Okay, let me go through the steps again to make sure I didn't miss
    anything. The user asked for the quotient of the area of
    Serengeti National Park in hectares divided by the birth year of
    the scientist who discovered oxygen. ...
```

Assistant Final Answer.

```
The quotient of the area of Serengeti National Park (1,500,000
    hectares) divided by the birth year of Joseph Priestley (1733) is
     **865.55**.

$$
\frac{1,\!500,\!000}{1733} \approx 865.55
$$

**Answer:** $\boxed{865.55}$
```