PATH-GRPO: PLACEMENT-AWARE TEACHER HINTING FOR DATA-SCARCE MULTI-TURN AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training a language-model agent with reinforcement learning (RL) is hard when training tasks are few and most attempts fail. With so few tasks, a supervised warm start before RL has little to learn from. Group-relative methods such as GRPO learn from reward differences within a group of attempts, so a group in which every attempt fails gives no learning signal. Most existing hint methods for such groups study single-turn reasoning, where the hint can only go in the prompt. In a multi-turn agent, a hint can enter at any turn, so where it enters becomes a design choice. We introduce PATH-GRPO (**P**lacement-**A**ware **T**eacher **H**inting with **GRPO**). When every attempt fails, a teacher chooses a restart turn, reads the reference solution, and writes an *insight* that explains how to find the answer. The policy retries from that turn, and we remove the insight before the update (*stripping*), so that the update sees the same inputs as evaluation, where the policy gets no help. With only 90 AppWorld tasks and 45 TravelPlanner queries, PATH-GRPO improves over GRPO at both 4B and 9B, on AppWorld Test-Challenge scenario goal completion ($+13.2$ and $+9.2$ points) and TravelPlanner leaderboard test success ($+29.1$ and $+4.5$). Our extensive ablations show that placement matters: on the main metrics, placing the insight at a teacher-chosen restart turn scores higher than placing it in the prompt or at a random turn. For these turn-level retries, both stripping the insight and letting the insight-generator teacher read the reference solution also yield higher scores on these metrics.

1 INTRODUCTION

An LLM agent does more than answer a question: it chooses tools, uses what they return, and keeps acting until the task is done. Benchmarks such as AppWorld (Trivedi et al., 2024) and TravelPlanner (Xie et al., 2024) check whether these interactions succeed. The harder question is how to train the agent's policy when most attempts fail and training data is scarce. We study this question with only **90** training tasks in AppWorld and **45** queries in TravelPlanner. With many demonstrations or successful attempts, a common recipe is supervised fine-tuning (SFT) or rejection-sampling fine-tuning (RFT) followed by Group Relative Policy Optimization (GRPO) (Shao et al., 2024). With so few tasks, these warm starts have little to learn from: on TravelPlanner, the base policy succeeds only once in 2,880 attempts (Section 4.2).

Without a useful warm start, GRPO must learn from the policy's own attempts, and most of them fail. GRPO compares several attempts at the same task and rewards those that do better than the group average. If every attempt fails, none is better than the others, and the task contributes no gradient. We call this a *dead group*. In the first training step of our AppWorld run (with `Qwen3-4B-Instruct-2507`), all eight initial attempts fail in 77.8% of groups (Figure 1, left). Such a group does not show that the task is impossible, only that no sampled attempt succeeded.

One way to create the missing reward difference is to guide another attempt at the task. We say a dead group is *rescued* when such a guided retry succeeds. NuRL (Chen et al., 2026) supplies abstract hints derived from the final answer, including hints from a stronger teacher, and LTE (Tang et al., 2026) lists the policy's earlier wrong answers as a hint. Both were designed for single-turn reasoning, where the hint can only be added to the prompt. Methods for multi-turn agents also use guided retries, with external help or with restarts from a chosen turn (Da et al., 2025; Shi et al., 2026; Liu et al., 2026). In a multi-turn agent, the teacher's guidance, which we call an *insight*, can enter at any turn, and this raises two questions. Where should it enter: at the start of a new attempt (*global placement*), at a later turn after the earlier actions are replayed (*turn-level placement*), or

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.

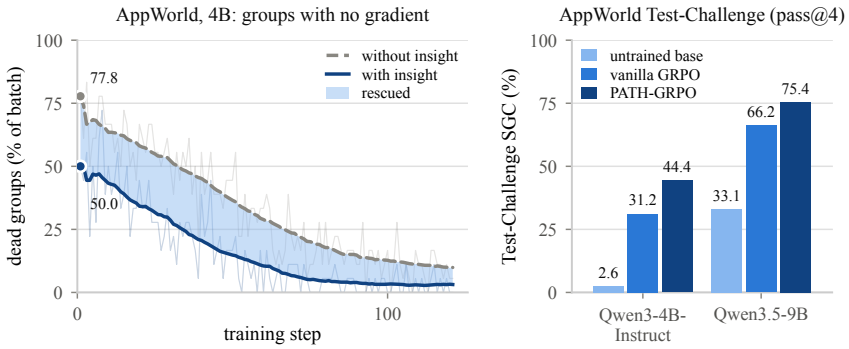

Figure 1: Dead groups and scenario goal completion (SGC) on AppWorld. *Left:* Share of dead groups during one PATH-GRPO run, with 18 groups per step and 8 unguided attempts per group before any retry. The "with insight" curve counts groups that stay dead after their guided retries. The shaded band shows the rescued groups. *Right:* Test-Challenge SGC from Table 1.

both (*stacked placement*)? And should it stay in the history used to calculate the training loss, even though the policy will see no insight at evaluation?

We introduce PATH-GRPO, which answers both questions within one training procedure (Section 3). When all initial attempts at a task fail, a teacher with fixed weights reads the failed interaction and its verifier feedback, and chooses a *restart turn* for each retry. In a separate call, the teacher reads the reference solution and writes an insight that explains how to find the correct result. The policy retries from the restart turn with the insight. Like LTE, we then remove the insight before the update, a step we call *stripping*, so that the update sees what the policy will see at evaluation. For example, if every attempt at a Venmo task requests money from all lunch attendees, including those who already paid, a turn-level insight at that step can say: "Before sending requests, check which attendees have already paid you." It names no person, amount, or API. Appendix C shows real insights logged during training. Our central finding is that in a multi-turn agent, where the insight enters and whether it stays in the update strongly affect how much the policy learns.

We train 4B and 9B models on both benchmarks. Our primary metrics are scenario goal completion (SGC) on AppWorld Test-Challenge and final success on TravelPlanner. The study makes three main contributions:

- **Teacher-guided RL for data-scarce multi-turn agents.** We train PATH-GRPO from public checkpoints, with no supervised stage. One guidance configuration outperforms GRPO, supervised warm starts followed by GRPO, and our adaptations of NuRL and LTE on the held-out tests of both benchmarks at both scales (Table 1). At 4B, a smaller Claude Haiku 4.5 teacher keeps most of the improvement over GRPO (Table 2).
- **Where and how the teacher should intervene.** In extensive ablations under the same stripping procedure, turn-level placement scores higher than global or stacked placement on both primary metrics. A teacher-chosen restart turn also scores higher than a random or midpoint turn, and for turn-level retries, stripping scores higher than keeping the insight. On AppWorld, global and turn-level insights rescue similar shares of dead groups, yet lead to very different final scores (Section 4.3, Table 2).
- **Insights from the reference solution.** NuRL and LTE build hints from a final answer or from the policy's own mistakes. In our multi-turn tasks, insights written from the full reference solution score higher, by 2.5 points on AppWorld and 10.2 points on TravelPlanner (Table 2).

2 BACKGROUND

Learning from a group of attempts. GRPO (Shao et al., 2024) samples K trajectories for the same task x and compares their verifier rewards $r_i = R(\tau_i)$. It assigns each trajectory the advantage

$$A_i = \frac{r_i - \text{mean}(r_{1:K})}{\text{std}(r_{1:K}) + \delta}, \tag{1}$$

where std is the sample standard deviation and δ keeps the denominator positive. A positive advantage favors the trajectory's assistant tokens, while a negative advantage discourages them. If

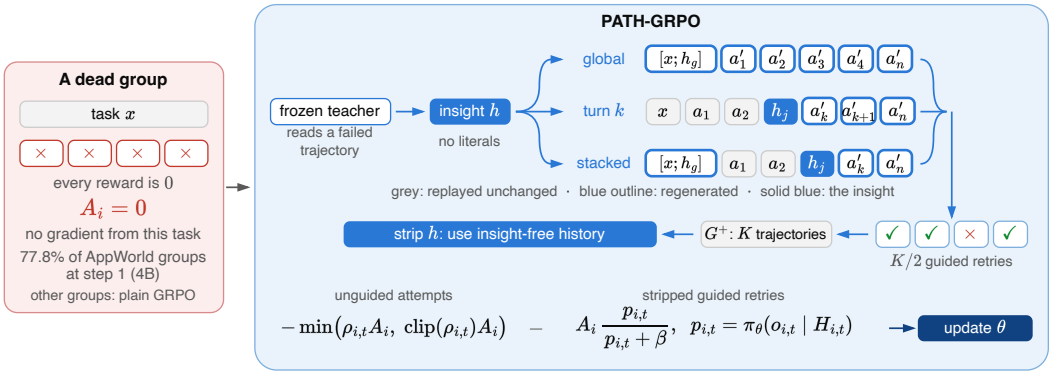

Figure 2: The three placement strategies in one training cycle, assuming that stripping succeeds. The label "no literals" refers to the writer prompt, which forbids copying solution names, fields, or values. Section 3.3 explains the update and what happens when stripping fails.

all rewards are equal, every advantage is zero. A dead group therefore supplies no reward-driven update. A single successful attempt is enough to give the group the reward difference it lacked.

Resuming an interaction. A trajectory $\tau = (a_1, u_1, \ldots, a_n, u_n)$ contains assistant turns a_s and the environment observations u_s that follow them. To resume before turn k, we re-execute the earlier actions to restore the environment, retain their original messages, and generate new turns from k onward. We write $o_{i,t}$ for token t of trajectory i (an assistant token) and $H_{i,t}$ for the history before that token. This history contains the task prompt, all preceding assistant text, and all observations, in their original order. It also contains the earlier text of the current turn. The observations help predict later actions, but they are not training targets because they come from the environment, not from the policy.

3 PATH-GRPO

PATH-GRPO gives the policy another chance to solve a task when its initial attempts all fail. The teacher helps it generate these retries, but our goal is a policy that can later act without that help. We therefore specify both where the insight enters the interaction and how the resulting trajectories contribute to training. Figure 2 shows the placement choices, and Algorithm 1 summarizes one training cycle.

3.1 WHEN AND HOW TO REQUEST AN INSIGHT

For each task, we reserve a total budget of K trajectories. We first sample $K/2$ of them from $\pi_{\theta_{\text{old}}}$, without guidance. We score these attempts with the binary verifier R. If at least one succeeds, we sample the other half without guidance as well. If all fail, we use teacher guidance for the remaining attempts. We keep all K trajectories in the final group $G^+(x)$, including failed retries. A successful retry can then receive a higher reward than the original failures. When a teacher call fails or provides no usable guidance, we sample a fresh attempt without an insight in place of that retry.

For turn-level retries, the teacher makes two separate calls. In the first call, the *selector*, the teacher reads the task, the failed interaction, and the verifier feedback. We ask it to return the turn where the agent made the commitment that caused the failure: the action or decision that locked the agent onto a path where the failed checks cannot be satisfied. The selector does not see the reference solution. In the second call, the *writer*, the teacher also receives the reference solution. We ask it to explain how the policy should find the correct result at the restart turn. The resulting insight should identify the wrong subgoal and give a procedure for inspecting or comparing the policy's own observations. The prompt forbids copying names, fields, or values from the solution. An insight that states the answer would let the policy copy it without learning how to find it, much as a student who is given the answer cannot solve the problem alone at test time. On TravelPlanner, both calls also receive the candidate pool: the benchmark's searchable data for the trip, such as flights, hotels, restaurants, and attractions. We ablate both choices in Section 4.3: a selector that also reads the reference solution, and a writer that does not (Appendix B.2). Appendix K gives the full prompts.

3.2 WHERE THE INSIGHT IS PLACED

An insight can guide a fresh attempt or a retry that preserves part of the failed interaction. We compare three strategies:

- **Global.** The teacher diagnoses the failed attempts and combines its diagnoses into one shared insight h_g. We add h_g to the task prompt. The policy then generates a new trajectory from the start. We write $[x; h_g]$ for this injection. The notation does not fix the order of the messages.
- **Turn-level.** The teacher writes a separate insight h_j for a restart turn k_j. We replay the earlier actions to restore the environment, keep their original messages, and insert the insight before generating the remaining turns. The replayed turns still count toward the maximum number of turns per attempt, like the original rollouts.
- **Stacked.** We provide both the global insight and a separate turn-level insight. The policy sees both insights and generates the turns from the restart turn onward. The replayed earlier turns stay fixed.

Our guidance configuration asks the teacher to choose the restart turn. The ablations instead pick a random turn or the midpoint turn of the failed trajectory. Each placement uses its own diagnosis, so the insight text differs between placements. We therefore compare complete strategies. Appendix K.1 gives the turn-index rules.

3.3 Mixed-Policy Optimization with Stripping

The final group contains attempts generated with and without guidance. We calculate their advantages together using Eq. 1, so they share the same reward baseline. However, they were sampled under different conditions. We train on both kinds of trajectories with mixed-policy optimization, following LUFFY (Yan et al., 2025) and LTE (Tang et al., 2026).

Unguided attempts. For a trajectory generated without guidance, the probability ratio is $\rho_{i,t} = \pi_\theta(o_{i,t} \mid H_{i,t}) / \pi_{\theta_{\text{old}}}(o_{i,t} \mid H_{i,t})$. Both probabilities use the full history defined in Section 2, including tool observations. We bound the log ratio before exponentiating, $\bar{\rho}_{i,t} = \exp(\text{clip}(\log \rho_{i,t}, -20, 20))$, and apply

$$
\begin{aligned}
C_{i,t} &= \min(\bar{\rho}_{i,t} A_i, \text{clip}(\bar{\rho}_{i,t}, 1-\epsilon_{\text{lo}}, 1+\epsilon_{\text{hi}}) A_i)\,, \\
\mathcal{L}_{\text{GRPO},i,t} &= \begin{cases} -C_{i,t}, & A_i \geq 0, \\ -\max(C_{i,t}, cA_i), & A_i < 0. \end{cases}
\end{aligned} \tag{2}
$$

Here ϵ_{lo} and ϵ_{hi} control clipping, and the dual-clip bound $c = 3$ limits updates with negative advantage.

Removing the insight. Since the policy receives no insight at evaluation, we remove the inserted insight before computing the training loss for a guided retry. After stripping, the assistant text and the observations stay in the trajectory, and we score the same actions with the usual history $H_{i,t}$. We write $\widehat{H}_{i,t}$ for the history that training actually uses, with or without the insight. After stripping, a check searches this history for remaining copies of the insight and logs them. Appendix K.2 explains the checks.

Guided retries. Removing an insight changes the inputs used to calculate a token's probability, but it does not change how that token was sampled. We cannot treat the retry as if it had been generated without guidance. We therefore follow the mixed-policy approach of LTE (Tang et al., 2026) and use its shaping loss.

$$
\begin{aligned}
p_{i,t} &= \pi_\theta(o_{i,t} \mid \widehat{H}_{i,t}), \qquad \bar{p}_{i,t} = \exp(\text{clip}(\log p_{i,t}, -20, 20))\,, \\
\mathcal{L}_{\text{guide},i,t} &= -A_i \, \frac{\bar{p}_{i,t}}{\bar{p}_{i,t} + \beta}, \qquad \beta > 0.
\end{aligned} \tag{3}
$$

Like LUFFY and LTE, this loss sets the probability of the sampling policy to one, so it uses the token probability directly instead of an importance ratio. The true sampling probability was computed with the insight in the context, so a ratio against it would compare two different inputs and could take extreme values. Without a ratio near one, PPO clipping would also be one-sided: it would remove the gradient of negative-advantage tokens whose probability is below $1 - \epsilon_{\text{lo}}$, which is why LUFFY omits it. The shaping term instead keeps each token's weight below one and shifts the gradient toward tokens that are unlikely without the insight. These are the actions that only the insight made possible, so the update focuses on the behavior the policy still has to learn. The clip on $\log p_{i,t}$ prevents numerical underflow. A token whose log-probability falls below this range gets zero gradient, so the update does not push up tokens that the policy essentially cannot produce without the insight, for example text that only the insight could have supplied.

Combining the losses. Let $z_i = 1$ if trajectory i uses the guided loss after stripping and any fallback, and $z_i = 0$ otherwise. The mask $m_{i,t}$ selects assistant tokens, including replayed text and newly generated reasoning. It excludes observations, prompt text, and explicit insight tokens. A separate mask may also exclude an entire trajectory that ended abnormally. We count the selected tokens separately as $N_{\text{on}} = \sum_{i,t}(1 - z_i)m_{i,t}$ and $N_{\text{guide}} = \sum_{i,t} z_i m_{i,t}$, then minimize

$$\mathcal{L}(\theta) \;=\; \frac{\sum_{i,t}(1 - z_i)m_{i,t}\mathcal{L}_{\text{GRPO},i,t}}{\max(1, N_{\text{on}})} \;+\; \frac{\sum_{i,t} z_i m_{i,t}\mathcal{L}_{\text{guide},i,t}}{\max(1, N_{\text{guide}})}. \tag{4}$$

Each denominator counts the selected tokens in the whole collected batch and stays the same for every optimizer minibatch. If a loss has no selected tokens, it contributes zero. A retry whose insight cannot be found keeps it and uses Eq. 2 with $\widehat{H}_{i,t}$. Because such a retry was sampled with guidance, this loss is only an approximation for it. Eq. 4 does not show the reference-KL regularization term. Its mask covers the trajectories of both losses. We use a KL coefficient of 10^{-3} at 9B and zero at 4B. Appendix I gives sufficient conditions under which guidance raises the chance of sampling a success.

Algorithm 1 One training cycle of PATH-GRPO.

Input: policy π_θ, task batch $\mathcal{B}$, trajectory budget K (even), teacher $\mathcal{T}$, placement ℓ
Output: updated policy parameters θ

```
θ_old ← θ
for task x ∈ B do
    Sample and verify K/2 trajectories G(x) without insights
    if all trajectories in G(x) fail then
        Generate guidance from the failed attempts (Section 3.1)
        Generate K/2 retries using placement ℓ; if a teacher call gives no usable guidance, sample that
        retry without an insight
    else
        Generate K/2 further trajectories without insights
    Verify the new trajectories and add them to G(x) to form G+(x)
Strip the insight from each guided retry; if the insight cannot be found, use the clipped loss
Recompute token probabilities and advantages over each G+(x)
Count training tokens for each loss and update θ with Eq. 4
```

4 EXPERIMENTS

We first test whether training with insights improves the policy when it receives no help at evaluation. We then compare placements and update inputs to understand which choices support that improvement. Our primary metrics are AppWorld Test-Challenge SGC and TravelPlanner final success. Table 1 reports the held-out test splits of both domains and TravelPlanner validation. The TravelPlanner ablations use the validation split, because the official leaderboard limits submissions to its privately held-out test set.

4.1 SETUP

Models and teachers. We train Qwen3-4B-Instruct-2507 and Qwen3.5-9B. Claude Sonnet 4.6 supplies the insights, and we replace it with Claude Haiku 4.5 in the smaller-teacher ablation study. We call the teachers only for the dead groups. We send them only public benchmark material: the task, the failed interaction, and the verifier feedback, plus the reference solution when the teacher writes the insight. For TravelPlanner, we also send the benchmark's candidate pool.

Task pools and splits. AppWorld (Trivedi et al., 2024) requires the agent to write Python against 9 apps and 457 APIs. We use 90 training tasks from 30 scenarios and 57 development tasks. Test-Normal contains 168 tasks from 56 scenarios. Test-Challenge contains 417 tasks from 139 scenarios. Test-Challenge introduces unseen apps, and uses up to six apps per task rather than the three per task seen during training. We report task goal completion (TGC), which requires all tests for a task to pass, and scenario goal completion (SGC), which requires every task variant in the scenario to pass. TravelPlanner (Xie et al., 2024) instead asks the agent to construct a plan satisfying commonsense and hard constraints. We use its 45 training queries, 180 validation queries, and 1,000 held-out test queries. We use the validation split for checkpoint selection and ablations. The official leaderboard scores our test submissions.

Evaluation. During evaluation, every policy receives the task and interacts with the tools without a teacher or insights. On AppWorld and on TravelPlanner validation, we sample four attempts per task and report pass@4 (Appendix H). On TravelPlanner validation, we also report three fractions: delivery, commonsense constraints, and hard constraints. Delivery checks that a plan is returned. Commonsense constraints are implicit rules of a valid trip, such as a reasonable route and no repeated restaurants. Hard constraints are the user's explicit needs, such as budget and room type (Appendix H). Final success is stricter: it requires all constraints to be satisfied.

For TravelPlanner test, we submit one plan per query and report the leaderboard's final pass rate as pass@1. The leaderboard returns aggregate rates and grouped constraint summaries. It does not return the outcome of each query, which pass@4 needs. Our test submissions use the same checkpoints, selected on development data, and the same decoding settings as the validation rows. Appendix H gives the scoring details.

Training and selection. The 4B policy is an instruct model, and the 9B policy is a thinking model. Within each model and domain, all methods share the same optimizer, task pool, verifier, context, decoding settings, and rollout budget (Table 14 in Appendix H lists the RL hyperparameters). We select checkpoints, prompts, and supervised training epochs on development data only: by pass@4 task accuracy on AppWorld and by final success on TravelPlanner.

Baselines. We compare against the untrained Base model and GRPO to measure the gain from training and the further gain from insights. We also adapt NuRL and LTE to the same mixed-policy framework. NuRL uses a global teacher insight and keeps it in the training inputs. LTE uses a deterministic summary of the policy's own failed attempts, places it globally, and strips it before the update. It makes no teacher call. These baselines are matched multi-turn adaptations, because both methods were designed for single-turn reasoning; Appendix H details the LTE adaptation. Compared with PATH-GRPO, NuRL changes the placement and the update inputs, and LTE changes the placement and the source of guidance.

To test whether supervised training is enough, we include three supervised baselines. SFT trains on reference solutions. Rejection-sampling fine-tuning (RFT) trains on accepted trajectories from the base model. RFT-hinted trains on accepted guided trajectories after stripping. We also start GRPO from each supervised model. To collect accepted trajectories, we repeatedly sample groups of 16 attempts, remove duplicates, and cap the number of trajectories per task. On AppWorld, the reference programs use helper functions and data that the agent cannot access. We convert them directly into SFT examples by including synthetic environment observations. Appendix D gives the construction and dataset sizes.

4.2 MAIN RESULTS

Table 1 evaluates one guidance configuration of PATH-GRPO across both domains and scales. It uses turn-level placement and a selector that reads the failed trajectory and verifier feedback. Only the writer receives the reference solution, and we ask it to derive a procedure from that solution. The reference-free writer is a separate ablation in Table 2.

The trained policy improves without help at evaluation. PATH-GRPO improves held-out performance in both domains. At 4B, AppWorld Test-Challenge SGC rises from 31.2 with GRPO to 44.4 with PATH-GRPO, a gain of 13.2 points. At 9B, it rises from 66.2 to 75.4, a gain of 9.2 points. TravelPlanner test pass@1 rises from 27.1 to 56.2 at 4B and from 82.5 to 87.0 at 9B, gains of 29.1 and 4.5 points. TravelPlanner validation final success at pass@4 also improves over GRPO, by 32.1 points at 4B and 2.1 points at 9B. Validation and test scores use different splits and different numbers of attempts. Appendix A compares PATH-GRPO with current state-of-the-art results for AppWorld (Chen et al., 2025).

Single-turn guidance methods do not transfer reliably. NuRL is competitive on AppWorld, where its Test-Challenge SGC reaches 40.2 at 4B and 73.3 at 9B, compared with 44.4 and 75.4 for PATH-GRPO. On TravelPlanner test, NuRL and LTE reach 43.6 and 36.8 at 4B, above GRPO's 27.1. At 9B, they reach 70.9 and 63.6, below GRPO's 82.5. These mixed results motivate the placement and stripping comparisons in Section 4.3. At 9B, the base model already solves more tasks: only 27.8% of AppWorld groups are dead at the first step, compared with 77.8% at 4B, and the teacher is called on only 3% of AppWorld training steps (Appendix F). Fewer dead groups leave less room for guidance, which fits the smaller lead over the other guidance methods at 9B.

Table 1: Results without teacher guidance (%). TravelPlanner test uses leaderboard pass@1; the other columns use pass@4. TGC and SGC mean Task and Scenario Goal Completion. TravelPlanner reports delivery, commonsense and hard-constraint fractions, and final success. Bold and underline mark best and second-best per model; Δ compares with the best other arm. Dashes denote insufficient RFT data.

	AppWorld (pass@4)				TravelPlanner				
	Test-Normal		Test-Challenge		Validation (pass@4)				Test pass@1
Method	TGC	SGC	TGC	SGC	Deliv.	Comm.	Hard	Final	Final
Qwen3-4B-Instruct-2507									
Base	21.2	7.7	10.4	2.6	97.4	54.0	13.8	0.0	0.4
GRPO	69.3	53.6	46.5	31.2	**100.0**	94.7	69.1	34.2	27.1
SFT	0.0	0.0	0.1	0.0	**100.0**	86.9	15.8	5.9	3.6
SFT + GRPO	0.0	0.0	0.1	0.0	**100.0**	86.8	17.7	9.0	5.0
RFT	33.8	18.2	16.2	7.1	–	–	–	–	–
RFT + GRPO	68.0	54.4	41.3	29.7	–	–	–	–	–
RFT-hinted	37.8	23.9	21.6	10.7	93.4	78.4	37.5	11.5	4.1
RFT-hinted + GRPO	77.6	66.1	54.3	<u>41.2</u>	<u>99.7</u>	96.2	66.6	41.6	40.3
NuRL (Chen et al., 2026)	<u>78.1</u>	<u>66.4</u>	<u>55.1</u>	40.2	**100.0**	**97.2**	<u>74.3</u>	<u>53.4</u>	<u>43.6</u>
LTE (Tang et al., 2026)	67.6	54.5	45.3	30.5	**100.0**	95.5	69.2	43.2	36.8
PATH-GRPO (ours)	**81.4**	**69.6**	**59.0**	**44.4**	**100.0**	<u>97.0</u>	**84.6**	**66.3**	**56.2**
Δ vs. best other	+3.3	+3.2	+3.9	+3.2	0.0	−0.2	+10.3	+12.9	+12.6
Qwen3.5-9B									
Base	76.3	61.4	52.1	33.1	96.8	61.8	7.4	0.4	0.0
GRPO	92.1	88.5	80.1	66.2	<u>98.2</u>	**97.9**	<u>96.3</u>	<u>93.9</u>	<u>82.5</u>
SFT	0.0	0.0	0.3	0.0	**100.0**	88.5	18.5	8.4	5.0
SFT + GRPO	0.0	0.0	0.4	0.0	**100.0**	89.0	21.3	10.0	0.0
RFT	83.2	70.3	56.0	39.5	–	–	–	–	–
RFT + GRPO	89.1	83.4	70.7	54.0	–	–	–	–	–
RFT-hinted	83.5	76.2	66.3	48.8	78.0	62.1	37.0	9.2	3.1
RFT-hinted + GRPO	92.2	86.9	77.7	63.2	78.5	73.1	61.9	39.3	21.0
NuRL (Chen et al., 2026)	92.8	88.9	<u>83.6</u>	<u>73.3</u>	93.8	93.1	89.9	86.5	70.9
LTE (Tang et al., 2026)	<u>93.1</u>	<u>91.1</u>	80.5	68.7	97.2	96.8	93.9	89.0	63.6
PATH-GRPO (ours)	**93.8**	**91.7**	**85.3**	**75.4**	**100.0**	<u>97.1</u>	**96.6**	**96.0**	**87.0**
Δ vs. best other	+0.7	+0.6	+1.7	+2.1	0.0	−0.8	+0.3	+2.1	+4.5

PATH-GRPO still scores above GRPO, and its lead over the best other method on AppWorld is 0.6 to 2.1 points.

The supervised baselines remain below PATH-GRPO. With so few tasks, supervised warm starts have little data. TravelPlanner's base policy yields only one accepted trajectory in 2,880 collection attempts at each scale, leaving too little data for plain RFT. Training directly on reference solutions (SFT) stays far below RL in both domains, and RFT + GRPO stays below GRPO on AppWorld Test-Challenge SGC at both scales (29.7 versus 31.2, and 54.0 versus 66.2). PATH-GRPO uses the same references differently: its writer turns them into procedural insights, and PATH-GRPO has the highest score in every test column of Table 1. Adding GRPO after RFT-hinted helps at 4B: RFT-hinted + GRPO reaches 41.2 AppWorld Test-Challenge SGC, between GRPO and PATH-GRPO. At 9B, it reaches 63.2, below both GRPO and PATH-GRPO. On TravelPlanner test at 4B, it reaches 40.3, again between GRPO and PATH-GRPO. Appendix D gives the sizes of the guided datasets, the collection budgets, and how we build the reference SFT data.

4.3 WHICH DESIGN CHOICES MATTER?

The main results do not show where the insight should enter, whether it should stay in the update, or what the teacher should read. Table 2 compares these choices at 4B, and Appendix B gives all metrics. The placement runs share the teacher, verifier, and rollout budget, and the writer always reads the reference solution. The placement also changes what the writer diagnoses, so each comparison tests a placement together with the insight written for it.

Turn-level placement scores highest. With stripping in every run, turn-level placement scores above global and stacked placement in both domains. It also scores above the Global (NuRL variant)

row, which keeps a global insight in the update. On TravelPlanner, the Global run never improved and then fell to zero, and restarts with different seeds failed in the same way. This comparison therefore also reflects training stability (Appendix B.1, Table 4).

A teacher-chosen restart turn scores highest. Within turn-level placement, the teacher-chosen turn scores above a random turn and the midpoint turn. The advantage is large on TravelPlanner (12.7 points over the midpoint turn) and small on AppWorld (1.3 and 1.5 points). On TravelPlanner, the Random turn run fell to zero across restarts with different seeds (Appendix B.1, Table 4). A random turn can come after the choice that caused the failure, and the replayed prefix then keeps that error. This run rescues only 2.4% of its dead groups, against 13.8% for PATH-GRPO (Appendix C, Table 9).

Table 2: Placement, update, and teacher choices at 4B (pass@4, %). AppWorld shows Test-Challenge SGC, and TravelPlanner shows validation final success. The Global (NuRL variant) row repeats NuRL from Table 1. Bold marks the best value.

Setting	AppWorld	TravelPlanner
GRPO	31.2	34.2
PATH-GRPO	**44.4**	**66.3**
Placement (with stripping)		
Global	13.5	0.9
Stacked	23.3	63.3
Restart turn		
Random turn	43.1	0.4
Midpoint turn	42.9	53.6
Insight kept in the update		
Global (NuRL variant)	40.2	53.4
Turn-level, kept	41.7	36.4
Reference solution		
Reference-free writer	41.9	56.1
Reference-informed selector	35.6	61.7
Teacher size		
Haiku teacher	41.7	64.8

Stripping helps turn-level retries but not global ones. With turn-level placement, stripping scores higher than keeping the insight in both domains. With global placement, stripping instead lowers the scores, from 40.2 to 13.5 on AppWorld and from 53.4 to 0.9 on TravelPlanner. We hypothesize that a global insight shapes the whole retry, so after stripping, every token of the retry is trained without the insight that produced it. A turn-level insight shapes only the turns after the restart turn, which the teacher places 71% and 61% of the way through the trajectory on average (Appendix B.1).

The reference solution helps the writer but not the selector. The reference-informed writer scores higher than the reference-free writer, by 2.5 points on AppWorld and 10.2 points on TravelPlanner. The selector shows the opposite effect. The Reference-informed selector, which also reads the reference solution, scores 8.8 and 4.6 points lower. We hypothesize that this selector compares the failed trajectory with the reference solution and picks the turn where they diverge. In a multi-turn task, however, the agent can reach the correct result along many paths, so the first divergence from the reference need not be the turn that caused the failure. PATH-GRPO therefore does not give the reference solution to the selector. It gives the reference solution only to the writer, which uses it as extra context to explain how to find the correct result from the chosen turn. Both comparisons also change the prompt instructions (Appendix B.2, Tables 5 and 6).

A smaller teacher keeps most of the gain. With the Haiku teacher, PATH-GRPO trails the Sonnet run by 2.7 and 1.5 points. It still keeps 10.5 of the 13.2-point AppWorld gain over GRPO and 30.6 of the 32.1-point TravelPlanner gain (Appendix B.3, Table 7).

Rescue rates do not explain the placement gap. On AppWorld, global and turn-level insights rescue 52.6% and 53.3% of the groups they guide, yet the Global run scores 30.9 points lower. This fits the hypothesis above: after stripping, the policy is trained on global rescues that depended on the insight throughout (Appendix C, Table 9).

Teacher cost. These improvements require extra teacher work even though the rollout budget is fixed. Across the two model scales, teacher tokens equal 0.2% to 7.0% of the policy's tokens on AppWorld and 28.9% to 35.0% on TravelPlanner. Appendix F gives the accounting, including repeated API requests, and Appendix E shows how performance changes during training.

5 RELATED WORK

Hints for failed attempts. Several methods help the policy produce better attempts after failure. NuRL (Chen et al., 2026) supplies abstract hints derived from the final answer, including teacher-generated hints, and LTE (Tang et al., 2026) lists the policy's own wrong answers as a hint.

GOLF (Huang et al., 2026a) combines external critiques with the group's attempts to write refined attempts, and adds them to training as off-policy examples. Two recent methods also target dead groups. DAPO (Yu et al., 2025) drops these groups and samples other tasks. In our 4B AppWorld run, this would drop 77.8% of the groups at the first training step (Figure 1). HDPO (Ding, 2026) works on single-turn math. It gives the policy ground-truth information, keeps the policy's correct solutions, and distills these solutions back into the policy. We study how to adapt these approaches to multi-turn agents: where an insight should enter and how it should be written.

Guidance in multi-turn agents. Earlier agent methods already use guidance and chosen restart turns. Agent-RLVR (Da et al., 2025) collects guided software-engineering trajectories, and ActGuide-RL (Ji et al., 2026) adds action guidance to the task prompt and adapts how much it gives to the task difficulty. ToolAnchor (Liu et al., 2026) studies intervention rounds and trains replacement actions before GRPO. Fission-GRPO (Zhang et al., 2026b) generates recovery attempts from diagnostic feedback. R^{3}L (Shi et al., 2026) and PivoARL (Guo et al., 2026) select turns for self-guided retries. Unlike these self-guided retries, our insights come from a teacher that reads the reference solution, and we test where such guidance should enter and whether it should stay in the inputs to the update.

Learning from privileged information. Guided retries change what the policy generates. Other methods instead give richer feedback on trajectories it has already produced. For example, EviBack (Ma et al., 2026) adds teacher supervision to dead groups, and SDAR (Lu et al., 2026) adds token-level signals from a teacher that sees retrieved skills. Removing hints before learning is also an older idea: STaR (Zelikman et al., 2022) writes a rationale while it sees the answer, then trains on that rationale without the answer. Prefix-RFT (Huang et al., 2026b) studies a different choice: its demonstration prefix stays in the input, and the question is whether those tokens also count in the loss. Our comparison of keeping and stripping the insight changes the input itself. Appendix J discusses further related methods.

6 Discussion and Limitations

How to use a reference solution in a multi-turn agent. Our ablations suggest a simple recipe for turning a reference solution into a learning signal. First, the teacher locates the failing turn without the reference solution. Second, the teacher reads the reference solution and explains how to reach the correct result from that turn. Third, the insight enters the retry at that turn and is removed before the update. On both primary metrics, each of these choices scores higher than its alternative in Table 2.

General capability. Training with insights changes general capability less than GRPO does. On 5 benchmarks, PATH-GRPO-based policy changes the mean score from its base model by -0.3 points at 4B and $+1.2$ points at 9B, while GRPO changes it by -1.6 and -2.1 points (Appendix G).

Limitations. PATH-GRPO needs a verifier and a reference solution for each training task, and it calls an external teacher. These needs match the data-scarce setting we target: both benchmarks provide a reference solution for each of their few training tasks, and the teacher is called only on dead groups. With the smaller Haiku teacher, PATH-GRPO keeps most of its gain (Section 4.3). Finally, each training run takes about 48 hours on eight H200 GPUs, so we report one run per setting.

7 Conclusion

When training data is scarce, many GRPO groups of a multi-turn agent contain only failed attempts and give no learning signal. We introduced PATH-GRPO, which turns these dead groups into a learning signal. A frozen teacher picks the turn where the agent went wrong, then reads the reference solution and writes an insight. The policy retries from that turn with the insight, and the insight is removed before the update. With one guidance configuration, only 90 AppWorld and 45 TravelPlanner training tasks, and no supervised stage, PATH-GRPO outperforms GRPO, supervised warm starts followed by GRPO, and our adaptations of NuRL and LTE on the held-out tests of both benchmarks at both model scales. Our extensive ablations show that in a multi-turn agent, where the insight enters and whether it stays in the update strongly affect how much the policy learns. More broadly, our results suggest that a few reference solutions, too few to help as supervised data, can still guide RL when a teacher turns them into turn-level insights. Since the method needs only a verifier, a teacher model, and one reference solution per training task, we expect it to apply to other multi-turn agent tasks with little training data.

AI USE STATEMENT

A generative model is a component of the proposed method. PATH-GRPO queries a frozen teacher LLM (Claude Sonnet 4.6, and Claude Haiku 4.5 for the smaller-teacher ablation) to choose restart turns and to write the insights that guide retries of dead groups. The insights guide the sampled retries and are removed before the update. The RFT-hinted baseline also trains on retries guided by these insights. The teacher is never fine-tuned, adapted, or distilled, and it is queried only on dead groups. Section 4 and Appendix F give the models, decoding settings, and teacher calls. Section 3.3 describes how we remove the insight and check for remaining copies.

Beyond this role in the method, we used generative AI tools for the following tasks with required disclosure: giving feedback on whether the paper's claims match its tables, figures; and polishing the wording and notation of the analysis without changing its argument. We have not used generative AI tools for designing the method, implementing it, running or analyzing the experiments, or formulating the mathematical claims. Additionally, we used generative AI tools for tasks with recommended disclosure: writing boilerplate code, drafting and editing parts of the paper for readability, searching for and checking related work. We reviewed every AI-assisted contribution, and we take full responsibility for the final content of this work, including its text, claims, and artifacts.

ETHICS STATEMENT

Both benchmarks are public and run in simulated environments. AppWorld simulates a digital society, and TravelPlanner checks plans with a deterministic checker. No human subjects, personal data, customer data, or proprietary data are used anywhere in training or evaluation, and we release no new data. The teacher LLM is accessed as a hosted inference endpoint, and we send no benchmark content to that endpoint beyond the prompts described in Section 4.

REPRODUCIBILITY STATEMENT

We implement PATH-GRPO on the verl reinforcement-learning framework (Sheng et al., 2024) and use vLLM (Kwon et al., 2023) to run evaluation. Each training run uses 8 H200 GPUs for about 48 hours. Section 3 and Algorithm 1 specify the full training cycle: how we collect trajectories, when we call the teacher, how we place and strip insights, and the losses in Eqs. 1 to 4. Section 4 names the models, teachers, and baselines. Appendix H lists the training and evaluation settings (Table 14), the bootstrap estimator, the TravelPlanner scoring rules and leaderboard submission, and our LTE adaptation. Appendix K gives the teacher prompts, abridged where marked, the rules that map the selector's output to a restart turn, and how we check for copies of the insight after stripping. Appendix D describes how we build the supervised baselines, and Appendix F reports the teacher calls and tokens. Appendix I analyzes when guidance raises the chance of sampling a success.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5be69a584901a26c521c2b51e40a4c20-Abstract-Conference.html.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Justin Chih-Yao Chen, Becky Xiangyu Peng, Prafulla Kumar Choubey, Kung-Hsiang Huang, Jiaxin Zhang, Mohit Bansal, and Chien-Sheng Wu. Nudging the boundaries of LLM reasoning. In *International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10008034.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive LLM agents, 2025. URL https://arxiv.org/abs/2502.01600.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Jeff Da, Clinton Wang, Xiang Deng, Yuntao Ma, Nikhil Barhate, and Sean Hendryx. Agent-RLVR: Training software engineering agents via guidance and environment rewards, 2025. URL https://arxiv.org/abs/2506.11425.

Ken Ding. HDPO: Hybrid distillation policy optimization via privileged self-distillation, 2026. URL https://arxiv.org/abs/2603.23871.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (ReST) for language modeling, 2023. URL https://arxiv.org/abs/2308.08998.

Weiyang Guo, Zesheng Shi, Longhui Zhang, Zeen Zhu, Min Zhang, and Jing Li. Agent reinforcement learning via pivotal-aware self-feedback retry, 2026. URL https://arxiv.org/abs/2607.03702.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Lei Huang, Xiang Cheng, Chenxiao Zhao, Guobin Shen, Junjie Yang, Xiaocheng Feng, Yuxuan Gu, Xing Yu, and Bing Qin. Bootstrapping exploration with group-level natural language feedback in reinforcement learning, 2026a. URL https://arxiv.org/abs/2603.04597.

Zeyu Huang, Tianhao Cheng, Zihan Qiu, Zili Wang, Yinghui Xu, Edoardo M. Ponti, and Ivan Titov. Blending supervised and reinforcement fine-tuning with prefix sampling. In *International Conference on Machine Learning*, 2026b. URL https://icml.cc/virtual/2026/poster/64528.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. In *International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/64121.

Yuxiang Ji, Zengbin Wang, Yong Wang, Shidong Yang, Ziyu Ma, Guanhua Chen, Zonghua Sun, Liaoni Wu, and Xiangxiang Chu. Learning agentic policy from action guidance, 2026. URL https://arxiv.org/abs/2605.12004.

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, 2025. URL https://arxiv.org/abs/2503.09516.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention, 2023. URL https://arxiv.org/abs/2309.06180.

Yu Li, Shu Hong, and Tian Lan. Localizing credit at the divergence: Path-conditioned self-distillation for LLM reasoning, 2026. URL https://arxiv.org/abs/2606.15576.

Weiting Liu, Jieyi Bi, Wanqi Zhou, Jianfeng Feng, Yining Ma, Ai Han, and Wenlian Lu. ToolAnchor: Anchoring counterfactual context to boost agentic tool-use capability, 2026. URL https://arxiv.org/abs/2607.14145.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-zero-like training: A critical perspective, 2025. URL https://arxiv.org/abs/2503.20783.

Zhengxi Lu, Zhiyuan Yao, Zhuowen Han, Zi-Han Wang, Jinyang Wu, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Self-distilled agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2605.15155.

Xiao Ma, Zhiquan Hu, Yi Wei, Chenchen Zhao, Yijun Chen, Jicheng Zhao, Yuming Li, and Chuang Dai. EviBack: Search-agent reinforcement learning via evidence-constrained teacher backoff, 2026. URL https://arxiv.org/abs/2607.23955.

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 *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 79791–79821, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c66e1fcc9691aae706250638f36f681b-Abstract-Conference.html.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

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, 2024. URL https://arxiv.org/abs/2409.19256.

Weijie Shi, Yanxi Chen, Zexi Li, Xuchen Pan, Yuchang Sun, Jiajie Xu, Xiaofang Zhou, and Yaliang Li. R^3L: Reflect-then-retry reinforcement learning with language-guided exploration, pivotal credit, and positive amplification, 2026. URL https://arxiv.org/abs/2601.03715.

Chenming Tang, Hsiu-Yuan Huang, Weijie Liu, Clive Bai, Saiyong Yang, and Yunfang Wu. Do not step into the same river twice: Learning to reason from trial and error. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8018–8032. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.363. URL https://aclanthology.org/2026.acl-long.363/.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, and Honggang Qi. Skill-SD: Skill-conditioned self-distillation for multi-turn LLM agents, 2026a. URL https://arxiv.org/abs/2604.10674.

Xinyi Wang, Jinyi Han, Zishang Jiang, Tingyun Li, Jiaqing Liang, Sihang Jiang, Zhaoqian Dai, Shuguang Ma, Fei Yu, and Yanghua Xiao. Don't tell the answer, truly guide the reasoning during RL rollouts. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 3437–3455. Association for Computational Linguistics, 2026b. doi: 10.18653/v1/2026.findings-acl.169. URL https://aclanthology.org/2026.findings-acl.169/.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, volume 37, pp. 95266–95290. Curran Associates, Inc., 2024. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html.

Zi-Han Wang, Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Jie Wu, Zhengzhou Cai, Yueqing Sun, Ziang Ye, Linji Hao, Qi Gu, Xunliang Cai, Yongliang Shen, and Yujiu Yang. AgentOPSD: Recursive self-distillation for agentic reinforcement learning, 2026c. URL https://arxiv.org/abs/2608.05987.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. SWE-RL: Advancing LLM reasoning via reinforcement learning on open software evolution. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 78500–78525. Curran Associates, Inc., 2025. doi: 10.52202/085713-2629. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/7107d4d2e837bde2171c6b71b5bde954-Abstract-Conference.html.

Haoze Wu, Chuqiao Kuang, Tianyi Zhuang, and Xiaoguang Li. Beyond outcome rewards: Step-level self-distilled policy optimization for deep search agents, 2026. URL https://arxiv.org/abs/2608.12764.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A benchmark for real-world planning with language agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54590–54613. PMLR, 2024. URL https://proceedings.mlr.press/v235/xie24j.html.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance, 2025. URL https://arxiv.org/abs/2504.14945.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 113222–113244. Curran Associates, Inc., 2025. doi: 10.52202/085713-3775. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488. Curran Associates, Inc., 2022. doi: 10.52202/068431-1126. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract.html.

Ranxu Zhang, Guinan Chen, Chenshaodong, Jinghao Lin, Xiaozhou Xu, Sunzhe, Yanyong Zhang, and Chao Wang. Agentic reinforcement learning with self-distilled reward shaping, 2026a. URL https://arxiv.org/abs/2608.03223.

Zhiwei Zhang, Fei Zhao, Rui Wang, Zezhong Wang, Bin Liang, Jiakang Wang, Yao Hu, Shaosheng Cao, and Kam-Fai Wong. Robust tool use via Fission-GRPO: Learning to recover from execution errors. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 40477–40491. Association for Computational Linguistics,

2026b. doi: 10.18653/v1/2026.acl-long.1880. URL https://aclanthology.org/2026.acl-long.1880/.

Binbin Zheng, Zijun Xie, Guanqun Zhao, Enlei Gong, Xing Ma, Xiaoliang Fu, and Zeyu Chen. Group-reflective self-distillation for agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2607.28076.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62178–62209. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhou24t.html.

Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. SWEET-RL: Training multi-turn LLM agents on collaborative reasoning tasks, 2025. URL https://arxiv.org/abs/2503.15478.

APPENDIX

A COMPARISON WITH PUBLISHED APPWORLD RESULTS

Table 3 compares PATH-GRPO with fourteen systems from Chen et al. (2025): four prompted models, two preference-optimization methods, and eight reinforcement-learning methods, including four LOOP variants. Here we report our method at pass@1 to match the single attempt of the published results.

Table 3: AppWorld single-attempt results (%). Rows above *Ours* are quoted from Chen et al. (2025). TGC = Task Goal Completion; SGC = Scenario Goal Completion. Bold marks the best value, and underlining marks the second best.

		Test-Normal		Test-Challenge	
System	Params	TGC	SGC	TGC	SGC
Prompted, no fine-tuning					
GPT-4o	—	48.8	32.1	30.2	13.0
OpenAI o1	—	61.9	41.1	36.7	19.4
Llama 3 70B	70B	24.4	17.9	7.0	4.3
Qwen2.5-32B-Instruct	32B	39.2	18.6	21.0	7.5
Preference optimization					
DPO-MCTS	32B	57.0	31.8	31.8	13.7
DMPO	32B	59.0	36.6	36.3	18.4
Reinforcement learning					
PPO (learned critic)	32B	50.8	28.9	26.4	10.5
RLOO	32B	57.2	35.7	36.7	17.4
GRPO	32B	58.0	36.8	39.5	22.4
GRPO, no KL	32B	59.0	35.7	42.7	21.3
LOOP (bandit)	32B	53.3	33.6	27.7	13.0
LOOP (turn)	32B	64.1	43.5	40.8	26.5
LOOP RwNorm (token)	32B	61.9	44.1	39.8	20.4
LOOP (token)	32B	<u>71.3</u>	<u>53.6</u>	<u>45.7</u>	<u>26.6</u>
Ours					
PATH-GRPO, Qwen3-4B-Instruct-2507	4B	63.5	44.3	39.5	20.8
PATH-GRPO, Qwen3.5-9B	9B	**85.2**	**74.0**	**69.8**	**50.6**

At 9B, PATH-GRPO exceeds all fourteen listed systems on all four metrics. It has roughly a quarter of the parameters of the fine-tuned systems. Its margins over LOOP (token), the strongest comparator in this table, are 13.9 on Test-Normal TGC, 20.4 on Test-Normal SGC, 24.1 on Test-Challenge TGC, and 24.0 on Test-Challenge SGC. The largest margins occur on Test-Challenge, which introduces unseen apps and is the hardest split.

At 4B, with an eighth of the parameters of the fine-tuned comparators, PATH-GRPO exceeds both preference-optimization systems, PPO, RLOO, and LOOP (bandit) on all four metrics. It exceeds LOOP RwNorm on three of the four metrics, and LOOP (turn) only on Test-Normal SGC (44.3 versus 43.5). LOOP (token) is ahead of the 4B PATH-GRPO on all four metrics, by 5.8 to 9.3 points.

Against the published 32B GRPO, the 4B PATH-GRPO gives mixed results. It leads on Test-Normal TGC (63.5 versus 58.0) and SGC (44.3 versus 36.8), ties on Test-Challenge TGC at 39.5, and trails on Test-Challenge SGC (20.8 versus 22.4). The no-KL GRPO row is also ahead of the 4B PATH-GRPO on both Test-Challenge metrics. The 4B PATH-GRPO also exceeds prompted OpenAI o1, by 1.4 to 3.2 points.

B Additional Ablation Studies

B.1 Placement, Turn Selection, and Stripping

Table 4 gives all metrics for the placement, turn-selection, and stripping comparisons summarized in Table 2. On average, the teacher places the insight at turn 14.2 on AppWorld and turn 10.2 on TravelPlanner, which is 71% and 61% of the way through the trajectory.

With stripping, the development task accuracy of the Global and Stacked runs peaked at 66.1% and 86.0%, against 95.4% for PATH-GRPO.

On TravelPlanner, the Global and Random turn arms never rose above their early final success (0.9% and 0.4%), and then stayed at zero until the runs ended. Several restarts with different seeds failed in the same way. We hypothesize that both arms fail because the policy never gets a success that it can reproduce without help. On TravelPlanner, every initial group is dead at the first step (Table 11), so early positive rewards come only from guided retries. When such retries are rare, or depend on the insight throughout, the update mostly lowers the probability of the policy's own failed attempts. Section 4.3 gives two observations that support this. The Global (NuRL variant) arm, which differs from the Global arm only in keeping the insight during the update, reaches 53.4 final success instead of 0.9. The Random turn arm rescues only 2.4% of its dead groups, the lowest rate of any TravelPlanner arm (Table 9).

The "Turn-level, kept" arm keeps the insight in the context used to compute the training loss. Otherwise, it uses the same training setup as PATH-GRPO. Trained for roughly twice as long, it reaches 41.7 SGC, still below the 44.4 of PATH-GRPO. These results favor stripping for turn-level guidance, even though the kept arm used more training compute.

Table 4: Placement, turn selection, and stripping on Qwen3-4B-Instruct-2507 (pass@4, %). Each row changes the factor named in its group heading. Metrics as in Table 1. Training-window differences are described in the text.

	AppWorld				TravelPlanner validation			
	Test-Normal		Test-Challenge		constraint fraction			
Arm	TGC	SGC	TGC	SGC	Deliv.	Comm.	Hard	Final
PATH-GRPO (ours)	81.4	69.6	59.0	44.4	100.0	97.0	84.6	66.3
Placement (with stripping)								
Global	47.1	32.3	24.7	13.5	99.1	55.2	12.4	0.9
Stacked	65.1	51.1	39.4	23.3	99.6	97.5	83.5	63.3
Restart turn								
Random turn	79.6	67.6	60.9	43.1	97.6	54.5	13.4	0.4
Midpoint turn	79.2	67.0	56.7	42.9	100.0	95.8	71.4	53.6
Insight kept in the update								
Turn-level, kept	74.1	64.9	57.7	41.7	100.0	93.1	60.8	36.4

B.2 What the Teacher May Read

Because the teacher selects the turn and writes the insight in separate calls, we can vary what each call reads and how it is instructed. We report two ablations on Qwen3-4B-Instruct-2507. Both keep the placement, the teacher model, the rollout budget, and the verifier fixed, and both use the AppWorld test and TravelPlanner validation metrics of Table 1 at pass@4.

What the selector may read. Table 5 compares three selector configurations. The trajectory-only selector reads the failed trajectory. The main selector also reads verifier feedback, while the third selector receives the trajectory, feedback, and reference solution. On TravelPlanner, the two selectors that read the feedback also receive the candidate pool. All three use the same reference-informed writer. Because the writer reads the reference solution, an insight can use it even when the selector does not.

Compared with the main selector, the trajectory-only selector loses 1.2 points of Test-Challenge SGC and 0.6 points of TravelPlanner final success, but gains 1.6 points of Test-Challenge TGC. The selector that also reads the reference solution loses 8.8 points of Test-Challenge SGC and 4.6 points of TravelPlanner final success, and it is the worst in six of the eight columns. As Section 4.3 discusses, a selector that reads the reference solution may pick the turn where the failed trajectory first diverges from it, and in a multi-turn task this need not be the turn that caused the failure.

Table 5: Selector ablation on Qwen3-4B-Instruct-2507 (pass@4, %). The writer's inputs stay fixed. Metrics as in Table 1. Bold marks the best value, and underlining marks the second best.

	AppWorld				TravelPlanner validation			
	Test-Normal		Test-Challenge		constraint fraction			
Selector's view	TGC	SGC	TGC	SGC	Deliv.	Comm.	Hard	Final
Trajectory only	<u>81.0</u>	<u>62.6</u>	**60.6**	<u>43.2</u>	**100.0**	<u>97.4</u>	<u>83.9</u>	<u>65.7</u>
PATH-GRPO (trajectory + feedback)	**81.4**	**69.6**	<u>59.0</u>	**44.4**	**100.0**	97.0	**84.6**	**66.3**
Trajectory + feedback + reference	77.8	62.1	52.1	35.6	**100.0**	**97.8**	81.5	61.7

What the insight writer may read. Table 6 compares a reference-free writer with the reference-informed writer used by PATH-GRPO. Both writers receive the failed trajectory and verifier feedback, but only the reference-informed writer receives the reference solution. We ask this writer for a selection or procedure rule that lets the agent find the right result from its own observations, without copying names, fields, or values from the solution.

On Test-Challenge, the policy trained with the reference-informed writer reaches 59.0 TGC and 44.4 SGC, compared with 57.8 and 41.9 when trained with the reference-free writer. The reference-informed configuration also leads in every TravelPlanner column, and its final success is 10.2 points higher. On Test-Normal, however, the policy trained with the reference-free writer scores higher: 84.5 TGC and 75.4 SGC, compared with 81.4 and 69.6 for the reference-informed writer.

The reference-free writer also produces shorter TravelPlanner insights (55 versus 452 tokens). With these shorter insights, the policy rescues a smaller share of dead groups: 3.5%, against 13.8% for the reference-informed writer (Appendix C). The larger TravelPlanner differences fit the idea that more detailed guidance helps the policy satisfy several constraints at once. We selected the reference-informed writer on development data and use it in both domains and at both scales. The price of this fixed choice is its lower Test-Normal score.

Table 6: Writer ablation on Qwen3-4B-Instruct-2507 (pass@4, %). Only the reference-informed writer reads the reference solution, and its prompt forbids copying literals from it. Metrics as in Table 1. Bold marks the better writer.

	AppWorld				TravelPlanner validation			
	Test-Normal		Test-Challenge		constraint fraction			
Writer	TGC	SGC	TGC	SGC	Deliv.	Comm.	Hard	Final
Reference-free writer	**84.5**	**75.4**	57.8	41.9	86.8	84.4	68.6	56.1
PATH-GRPO (reference-informed)	81.4	69.6	**59.0**	**44.4**	**100.0**	**97.0**	**84.6**	**66.3**

B.3 EFFECT OF A SMALLER TEACHER

Table 7 tests whether the gains depend on the default Claude Sonnet 4.6 teacher by replacing it with the smaller Claude Haiku 4.5. We make this replacement in both teacher calls: the selector and the writer. Both runs start from Qwen3-4B-Instruct-2507 and use the same placement, rollout budget, verifier, and teacher instructions.

The gain holds with the smaller teacher. Sonnet scores higher in seven of the eight columns, with its largest lead of 3.4 points on TravelPlanner validation hard constraints and a 2.7-point lead on Test-Challenge SGC. Haiku leads on Test-Normal SGC, at 70.0 against 69.6. With the Haiku teacher,

PATH-GRPO still scores higher than every 4B baseline in Table 1 on all four AppWorld metrics and on TravelPlanner validation hard constraints and final success. The Haiku run keeps 10.5 of the 13.2-point gain that the Sonnet run has over GRPO on Test-Challenge SGC, and 30.6 of the 32.1-point gain in TravelPlanner validation final success.

Table 7: Teacher ablation on Qwen3-4B-Instruct-2507 (pass@4, %). The Haiku teacher replaces Sonnet in both the selector and the writer. Metrics as in Table 1. Bold marks the better teacher.

	AppWorld				TravelPlanner validation			
	Test-Normal		Test-Challenge		constraint fraction			
Arm	TGC	SGC	TGC	SGC	Deliv.	Comm.	Hard	Final
PATH-GRPO (Sonnet teacher)	**81.4**	69.6	**59.0**	**44.4**	**100.0**	**97.0**	**84.6**	**66.3**
Haiku teacher	80.6	**70.0**	57.7	41.7	99.0	96.6	81.2	64.8

C INSIGHT EXAMPLES AND RESCUE RATES

We examine what the guidance says and how often guided retries rescue a dead group. For the ablation runs, Table 9 summarizes how long the insights are, how often guided retries rescue a dead group, and how often dead groups occur.

Paired insights address different errors. Figure 3 shows two insights for a failed bill-splitting task. The agent divided the bill by the wrong number of people and sent money instead of requesting it. The global insight, placed before turn 1, gives three rules and addresses the payment-direction error. The turn-level insight at turn 24 of 30 addresses the divisor, which the global rules omit. Neither excerpt names an API call, field, or answer value; the turn-level rule identifies who shares the bill without giving the divisor. The group was rescued.

Global insight, written before turn 1

Re-examine whether sending money to someone and requesting money from someone are the same operation: the task specifies one of these and the agent is consistently doing the other.

Re-check where the roommates' payment identifiers should come from; the agent keeps deriving them from phone contacts, but a more authoritative source may already hold them.

Confirm the actual current date and read the real bill amount from the appropriate source rather than assuming which file represents last month.

Turn-level insight, at turn 24 of 30

The bill is shared equally among your roommates *and you*: re-examine how many people the total should be divided by before calculating each person's share.

Figure 3: Paired insights for one failed AppWorld task at one step of a stacked run, condensed for readability. The excerpts address different errors, and the group was rescued.

Six additional cases include two failed rescues. Table 8 shows six further cases. The excerpts identify missed procedures without naming an exact call, field, or answer value. Two groups remain unrescued. In the case at turn 7, the insight points to "other apps or data sources". In the case at turn 3, it refers to "a different level of the data structure". These cases illustrate that procedural guidance can leave the policy without a successful retry.

Group rescue and final success measure different outcomes. We measure group rescue from the training logs, which hold one entry per training step. Within each domain, we use the same early window of training for every arm. Runs that stopped inside this window contribute the steps they completed, and a few log entries are missing.

Table 8: Six turn-level insights from one stacked AppWorld run on Qwen3-4B-Instruct-2507, ordered by restart turn and condensed for readability. "k/turns" gives the restart turn and the trajectory length. We chose rows to show both outcomes and a range of turns, so they do not give a rescue rate; Table 9 gives the measured rate.

k/turns	What the agent did wrong	Turn-level insight (condensed)	Outcome
3/8	Assumed that each song has its own like count; in this app, like counts belong to playlists, so every song showed zero likes and the maximum was arbitrary.	The playlists you retrieved contain song identifiers, but check whether individual songs have their own like count, or whether the count you need belongs to a different level of the data structure.	Not rescued
4/12	Read a play count from a field that the songs do not have, so the maximum defaulted to the first song in the album.	The album's song data does not include play counts, so are you sure "most listened to" can be determined from what is visible in that response, or does the API offer a way to reach private statistics about songs?	Rescued
6/7	Sorted ascending and read only the first page of results, so the highest play count was never seen.	The sort direction prefix you choose decides whether the highest or the lowest value appears first: are you sorting to get the top result at the front of the list?	Rescued
7/8	Found the supervisor's password API, decided that it did not apply to the phone app, and gave up.	Before giving up, consider whether other apps or data sources already hold your roommates' identities and the bill amount, and whether you have checked every information store the task points to.	Not rescued
8/9	Fetched every received transaction without filtering to the coworkers the task named.	The task asks about money received from coworkers specifically: are you accounting for who the senders are, or treating all received transactions the same?	Rescued
12/17	Summed every cable bill file it found, including the previous year's, without establishing what "this year" is.	Before narrowing down the files, make sure you have identified which bills actually belong to "this year", rather than treating all available files the same regardless of their date.	Rescued

Within each window, we sum group counts before computing the rates in Table 9. *Dead* divides the number of dead groups by the total number of initial groups. *Rescued* divides the number of groups with a successful guided retry by the number that received usable guidance. In these windows, every dead group received usable guidance, so the rescue rate is the same whether we divide by dead groups or by guided groups.

The other columns describe insight length and the policy's later performance. *Insight* is the mean insight length in estimated tokens (characters divided by four), measured on turn-level insights when an arm has them. The PATH-GRPO writer is asked to give guidance for each distinct wrong subgoal or slot (a field of a TravelPlanner plan, such as a meal, an attraction, or the accommodation). Among 930 turn insights from three TravelPlanner runs, about half contain one rule and the rest two to four. *Success* repeats each arm's reported score without guidance: Test-Challenge SGC on AppWorld and validation final success at pass@4 on TravelPlanner.

On AppWorld, the Global arm, which also strips the insight, rescues almost as many groups as PATH-GRPO (52.6% versus 53.3%). Yet its dead-group share is higher (45.2% versus 19.2%), and its Test-Challenge SGC is 30.9 points lower. On TravelPlanner, every initial group in the Global arm is dead, and guidance rescues 7.6% of them. So a successful guided retry does not by itself mean better performance when the policy acts without guidance.

Across the eight arms, the dead-group share has Spearman rank correlation -0.98 with success on AppWorld and -0.95 on TravelPlanner. The corresponding correlations between rescue rate and

success are -0.05 and 0.83. These are associations across eight arms whose training phases differ, not causes.

Table 9: Logged ablation summaries using the training windows and pooled counts defined in the text. Dead, Rescued, and Success are percentages; Insight is estimated tokens. Success uses AppWorld Test-Challenge and TravelPlanner validation.

	AppWorld				TravelPlanner			
Arm	Dead	Rescued	Insight	Success	Dead	Rescued	Insight	Success
PATH-GRPO (ours)	19.2	53.3	60	44.4	55.1	13.8	452	66.3
Placement (with stripping)								
Global	45.2	52.6	274	13.5	100.0	7.6	622	0.9
Stacked	36.2	50.0	64	23.3	58.4	25.7	381	63.3
Restart turn								
Random turn	19.6	47.4	62	43.1	100.0	2.4	418	0.4
Midpoint turn	20.5	47.3	61	42.9	72.7	9.9	388	53.6
Selector's view								
Trajectory only	20.1	48.7	61	43.2	61.5	16.8	465	65.7
Trajectory + feedback + reference	21.4	52.6	59	35.6	67.9	11.4	470	61.7
Writer								
Reference-free writer	20.7	41.3	51	41.9	78.0	3.5	55	56.1

Rescue rates across selector configurations. Across the three selector configurations, rescue rates range from 48.7% to 53.3% on AppWorld and from 11.4% to 16.8% on TravelPlanner (Table 9). Mean insight length barely changes across these configurations: 59 to 61 estimated tokens on AppWorld and 452 to 470 on TravelPlanner.

D CONSTRUCTION OF THE SUPERVISED BASELINES

The supervised baselines in Table 1 differ only in where their training data come from. Table 10 gives the collection and dataset sizes. Sampled arms use the same task pool, verifier, context length, and sampling settings as GRPO, and draw rollouts in groups of 16. An *accepted* rollout passes the full verifier. We select the supervised training epoch on the development split.

Table 10: Collection and dataset sizes of the supervised baselines. Accepted counts rollouts that pass the verifier, with the share in parentheses. Rows counts training samples after filtering, deduplication, and a per-task cap; Tasks counts the tasks they cover. For 9B TravelPlanner RFT-hinted, the dataset was built partway through collection, from a subset of the listed rollouts.

Domain	Arm	Scale	Rollouts	Accepted	Rows	Tasks
AppWorld	SFT	4B	0	—	90	90
AppWorld	SFT	9B	0	—	90	90
AppWorld	RFT	4B	6,000	595 (9.9%)	136	40
AppWorld	RFT	9B	6,000	2,610 (43.5%)	322	83
AppWorld	RFT-hinted	4B	6,816	813 (11.9%)	99	39
AppWorld	RFT-hinted	9B	6,912	3,147 (45.5%)	79	21
TravelPlanner	SFT	4B	0	—	45	45
TravelPlanner	SFT	9B	0	—	45	45
TravelPlanner	RFT	4B	2,880	1 (0.03%)	not buildable	
TravelPlanner	RFT	9B	2,880	1 (0.03%)	not buildable	
TravelPlanner	RFT-hinted	4B	25,472	226 (0.89%)	94	15
TravelPlanner	RFT-hinted	9B	61,264	146 (0.24%)	62	17

D.1 SFT ON REFERENCE SOLUTIONS

SFT trains on the benchmark reference solutions, following the `SFT-GT` arm of Chen et al. (2025). On AppWorld, each statement of a reference program becomes one assistant turn, which gives 90 samples with 4 to 17 turns each. The reference programs use helper routines and data that the agent cannot access, so we do not execute them. Instead, each turn receives a synthetic environment observation. TravelPlanner provides a reference plan for each query but no reference trajectory, so we construct one trajectory per plan: we derive the searches from the plan, record their real tool outputs, and end with the reference plan as the final answer. This gives 45 samples, one per training query.

D.2 RFT ON ACCEPTED ROLLOUTS

RFT samples rollouts from the untrained model and keeps the accepted ones, after removing duplicates and keeping at most the 4 shortest per task. On TravelPlanner, 2,880 rollouts give only one accepted trajectory at each scale, because final success requires every constraint to hold on the same plan. This is too little data to train on, so Table 1 shows dashes for these arms.

D.3 RFT-HINTED

For each dead group, RFT-hinted asks the teacher for an insight, collects new rollouts with it, and trains on the accepted ones after stripping the insight. It uses the same writer as PATH-GRPO. On TravelPlanner at 4B, insights raise acceptance from 0.03% to 0.89%. Higher acceptance does not always give more rows: hinted collection covers only dead groups, and the per-task cap (4 on AppWorld, 12 on TravelPlanner) limits each task. On AppWorld at 9B, for example, RFT-hinted keeps 79 rows over 21 tasks, while RFT keeps 322 rows over 83 tasks.

D.4 ADDING GRPO AFTER SUPERVISED TRAINING

For the + GRPO arms, we start GRPO from the best supervised checkpoint and use the same RL setup as GRPO from the base model.

E TRAINING DYNAMICS

Table 1 shows how well each trained policy performs, and Figure 4 shows when during training this performance is reached. We smooth the development scores with a centered average over nine evaluation points. The x-axis divides training steps by the step of GRPO's best smoothed score, so GRPO reaches its best at 1. The dashed line marks that score, and the arrow shows where PATH-GRPO first reaches it. The corner block gives each method's best smoothed score.

On the smoothed curves, PATH-GRPO reaches GRPO's best score before GRPO does, in both domains and at both scales. It reaches that score after 13% to 65% of the training steps that GRPO needs.

F TEACHER COST

We request guidance only for dead groups, so teacher use depends on how often they occur (Algorithm 1). To measure that use, Table 11 covers the training of each reported model and uses every step with a logged policy token count. For teacher tokens, we sum the input and output usage returned by the API. For policy tokens, we count the non-padding tokens in the assembled training batch. The call total counts each request once. It excludes repeated API requests. The token totals, however, include them. The dead-group column describes a different quantity: the fraction of initial groups that are dead at the first step, before any guidance.

On AppWorld, teacher tokens equal 7.0% of the policy tokens at 4B and 0.2% at 9B. On TravelPlanner, they equal 28.9% and 35.0%. These ratios compare token counts.

Relative to policy tokens, teacher use is largest early in these runs (Figure 5). As training continues, the cumulative ratio falls in all four settings: on AppWorld, by about one order of magnitude at 4B

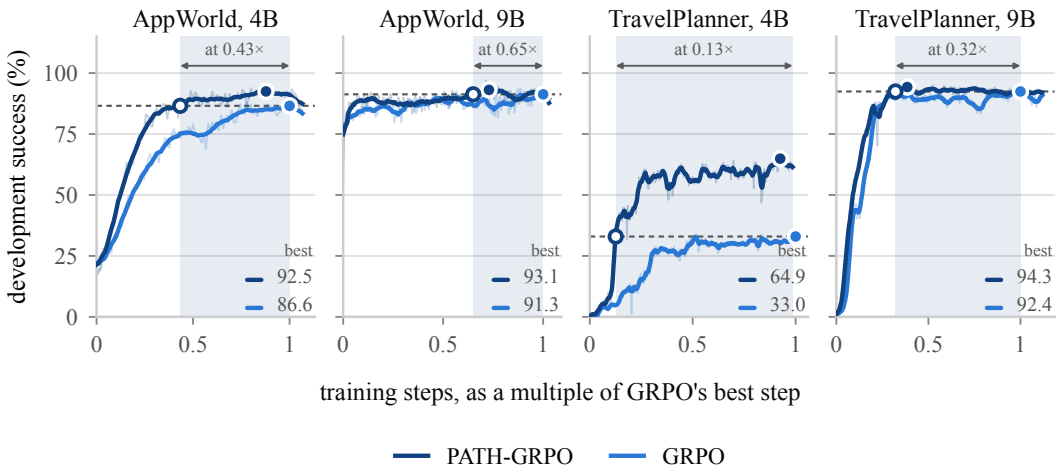

Figure 4: Development learning curves at pass@4. Solid curves are centered moving averages over nine evaluation points, and faint lines show the raw data. Both curves in a panel end at the same step.

Table 11: Teacher calls and tokens during the training of each reported model. Teacher tokens include input and output. The final column is a token ratio, not a cost ratio.

Domain	Model	Dead groups at step 1	Steps with a call	Teacher calls	Teacher tokens	Teacher / policy tokens
AppWorld	4B	77.8%	40.1%	12,606	172M	7.0%
AppWorld	9B	27.8%	3.1%	542	7M	0.2%
TravelPlanner	4B	100%	52.5%	53,966	898M	28.9%
TravelPlanner	9B	93.3%	57.1%	19,910	616M	35.0%

and two at 9B. Calls occur on 40% of the AppWorld steps at 4B and 3% at 9B. Over the last fifth of training, the mean per-step ratio is 0.21% at 4B and 0.00% at 9B. The teacher is called less often as fewer groups fail completely, but total teacher use still grows with training length.

On TravelPlanner, the teacher initially uses more tokens than the policy: the ratios start at 216% at 4B and 110% at 9B. At that stage, nearly every initial group is dead, and the teacher reads the failed trajectories when asked for guidance. This high early use keeps the cumulative ratios of 28.9% and 35.0% above the per-step ratios seen late in training.

Some API requests are sent several times. Including these repeats, the logs record 71,811 API requests on AppWorld at 4B and 2,530 at 9B, and 438,188 and 270,678 on TravelPlanner. In all four runs, the steps with a teacher call are exactly the steps with usable guidance.

G General-Capability Regression

To check whether TravelPlanner training changes other capabilities, we compare GRPO and PATH-GRPO policies with their untrained bases at both scales (Table 13). We keep the harness, prompt format, and decoding settings fixed within each scale. GRPO is trained the same way but without insights, so it serves as a comparison.

We test these capabilities with the public suites in Table 12: knowledge, instruction following, math, code, and function calling. We run the first four suites with the `lm-evaluation-harness` (Gao et al., 2023). For function calling, we use BFCL v4 (Patil et al., 2025) with its own harness. None of these suites is used for training.

At 4B, PATH-GRPO stays within a point of its base on instruction following, math, and code, gains 0.7 on function calling, and loses 3.8 on MMLU-Pro. Its five-suite mean change is -0.3, against GRPO's -1.6.

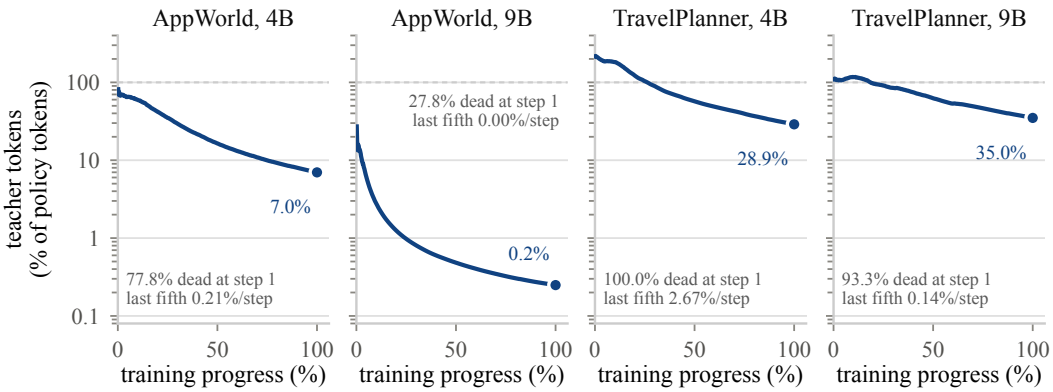

Figure 5: Cumulative teacher tokens as a percentage of policy tokens during the training of each reported model. Endpoints match Table 11; the dashed line marks equal token counts. Each panel also prints the share of dead groups at step 1 and the mean per-step ratio over the last fifth of training.

Table 12: Public regression suites for TravelPlanner-trained checkpoints.

Task	Capability tested
`mmlu_pro` (Wang et al., 2024)	knowledge and STEM reasoning
`ifeval` (Zhou et al., 2023)	instruction following
`gsm8k` (Cobbe et al., 2021)	grade-school math
`mbpp` (Austin et al., 2021)	Python code generation
`bfcl-v4` (Patil et al., 2025)	function calling

At 9B, the mean changes are $+1.2$ for PATH-GRPO and -2.1 for GRPO. Single suites can still drop: the largest drops for PATH-GRPO are 3.8 points on MMLU-Pro at 4B and 3.7 points on BFCL v4 at 9B. GRPO's mean change is below that of PATH-GRPO at both scales.

Table 13: General-capability results after TravelPlanner training (%). GRPO is trained the same way without insights. Δ is the mean change from the untrained base over the five suites in Table 12.

Model	Checkpoint	MMLU-Pro	IFEval	GSM8K	MBPP	BFCL v4	Δ vs base
Qwen3-4B-Instruct-2507	untrained base	67.5	83.4	91.5	69.0	35.2	—
	GRPO	66.9	83.7	90.8	64.2	33.0	-1.6
	PATH-GRPO (ours)	63.7	83.9	92.3	69.2	35.9	-0.3
Qwen3.5-9B	untrained base	75.9	83.9	95.5	69.2	48.4	—
	GRPO	76.8	85.2	95.1	56.4	48.9	-2.1
	PATH-GRPO (ours)	76.3	83.2	94.7	80.2	44.7	$+1.2$

H TRAINING AND EVALUATION PROTOCOL

Table 14 lists the reported RL settings. Policy sampling settings are shared by training and evaluation. Appendix D gives the supervised collection and dataset sizes.

Bootstrap estimation. We compute pass@4 with bootstrap resampling with replacement, using 1,000 resamples with seed 42 for each value. For a task x, let $\bar{r}_x$ be the mean reward over its 4 attempts, that is, the fraction that succeed. The expected bootstrap pass@4 for this task is $b_x = 1 - (1 - \bar{r}_x)^4$, which differs from the usual without-replacement pass@k estimator. TGC averages b_x over tasks, while SGC averages, over scenarios, the product of b_x over the tasks in each scenario. For scenario success, we resample each task variant independently and then combine them. The variation across resamples is not variance across training seeds, and it is not a confidence interval for the difference between two methods.

Table 14: Reported training and evaluation settings. Shared entries apply to both domains. Rows that name a scale apply only to that model. Token limits apply to the policy. The temperature and top-p values follow the decoding settings recommended by each model's official guidance.

Setting	AppWorld	TravelPlanner
Training tasks	90	45
Development split	57	180
Task batch / PPO batch	18 / 18	15 / 15
Maximum prompt tokens	4096	8192
Maximum response tokens	32768	32768
Maximum model context	65536	65536
Maximum turns	50	30
Optimizer	AdamW	
Learning rate	10^{-6}	
PPO epochs per collected batch	1	
Rollout group size	16	
Shaping constant β	0.1	
PPO clip widths (lower / upper)	0.2 / 0.28	
Dual-clip bound c	3	
Entropy coefficient	0	
4B temperature / top-p	0.7 / 0.8	
9B temperature / top-p	0.6 / 0.95	
Model type (4B / 9B)	instruct / thinking	
KL coefficient (4B / 9B)	0 / 10^{-3}	

Scoring TravelPlanner validation plans. TravelPlanner (Xie et al., 2024) checks three things. Delivery checks that the agent returns a final plan instead of looping, failing, or reaching the turn limit. The eight commonsense constraints check that the plan stays within the sandbox data, has complete information, keeps each day's activities in that day's city, follows a reasonable city route, repeats no restaurant or attraction, uses non-conflicting transport, and meets minimum-night stays. The hard constraints are the user's explicit needs in the query: budget, room rule, room type, cuisine, and transportation. Missing or unparseable plans receive zero delivery and constraint scores. The commonsense score is the fraction of the checker's commonsense checks that the plan passes. The checker scores hard constraints only for plans that pass the presence and sandbox-validity checks, and gives other plans a hard score of zero. It skips hard constraints that do not apply to the query. A valid plan with no applicable hard constraint therefore scores one, and a checker exception scores zero. Final success requires all commonsense and applicable hard checks on the same plan. For each query and metric, pass@4 is the average, over bootstrap resamples of four attempts, of the best score in each resample. So the best score in one column can come from a different attempt than in another column.

TravelPlanner leaderboard test. The 1,000-query test split does not release its constraint annotations, so we submit one plan per query to the official leaderboard.[1] The leaderboard counts the plans that satisfy all applicable commonsense and hard constraints and divides by 1,000. It also returns constraint summaries by difficulty and trip length, but not the result of each query, which pass@4 needs. We therefore report test final success at pass@1, without bootstrapping.

Deterministic LTE adaptation. For each failed attempt, the LTE adaptation builds a short digest: the exit reason, the submitted answer or last action, and the end of the final observation. On AppWorld the answer is the task-completion answer, when present; on TravelPlanner it is the submitted plan. We remove duplicate digests. The global guidance text then contains the digests, an instruction to be more concise if attempts were truncated, or both. This adaptation reads no reference solution or failed-check descriptions and makes no teacher call. It builds groups and strips this text in the same way as PATH-GRPO.

[1] https://huggingface.co/spaces/osunlp/TravelPlannerLeaderboard

I Formal Analysis: When Guidance Raises the Success Probability

We ask when guidance increases the probability of sampling a successful completion. We compare each guided rollout with an unaided rollout from the same starting context, and we give sufficient conditions for an increase. This analysis is related to the sampling analysis of LTE (Tang et al., 2026). These conditions describe how the sampling distributions would need to behave.

We fix the policy. We also condition on three things: the initial dead group $G(x)$ of size $K/2$, the restart turns, and the teacher outputs that were actually produced. We denote this information by $\mathcal{F}$. The second half of the group consists of $K/2$ appended rows. For appended row j, let ξ_j be its starting context: $\xi_j = x$ under global placement and $\xi_j = (x, \omega_{j,<k_j})$ under turn-level or stacked placement. Here $\omega_{j,<k_j} = (a_{j,1}, u_{j,1}, \ldots, a_{j,k_j-1}, u_{j,k_j-1})$ is the prefix that the retry keeps: the actions and observations before turn k_j. Global placement uses the injection operator $[x; h_g]$ defined in Section 3.2. Under stacked placement, the row receives both the global insight h_g and its turn-level insight h_j, and the prefix stays fixed.

Let P_j be the distribution over completions from ξ_j without guidance, let Q_j be the distribution with guidance, and let $\mathcal{C}_j$ be the set of correct completions from that context. A completion includes regenerated actions, environment observations, and termination. To compare the two distributions, consider the action probabilities at each regenerated turn s. Write these as $p_{j,s}(a_s \mid S_s)$ and $q_{j,s}(a_s \mid S_s, g_j)$, where $S_s = (\xi_j, a^{\text{new}}_{<s}, u^{\text{new}}_{<s})$ contains the interleaved history since the restart. Here g_j is the guidance that row j receives: $g_j = h_g$ under global placement, $g_j = h_j$ under turn-level placement, and $g_j = (h_g, h_j)$ under stacked placement.

Assumptions. For each appended row with usable guidance:

- (A1) *No decrease on correct paths.* On every correct completion with positive P_j probability, every regenerated action satisfies $q_{j,s}(a_s \mid S_s, g_j) \geq p_{j,s}(a_s \mid S_s)$. At least one regenerated action on one such completion satisfies a strict inequality.
- (A2) *Reachability.* $P_j(\mathcal{C}_j) > 0$. For a row that restarts at turn k_j, a correct continuation of the retained prefix must be possible within the remaining budget.
- (A3) *Common environment and regularity.* The guided and unaided distributions share three things: given ξ_j, the same distribution of starting environment states; at corresponding histories, the same environment transitions; and the same stopping and budget rules. Insights affect the policy's actions, not the environment directly. The set of complete trajectories is finite or countable, and the unaided probabilities in the ratios of the proof are positive on the paths we consider.

Proposition 1 (Conditional sampling gain)**.** *Under (A1)–(A3), row j has guided success probability $q_j = Q_j(\mathcal{C}_j)$ strictly larger than $p_j = P_j(\mathcal{C}_j) > 0$. This comparison holds separately for global, turn-level, and stacked placement, using each row's own starting context. Suppose at least one appended row has usable guidance and every such row satisfies these assumptions. We append $K' = K/2$ rows and use a fresh unguided rollout wherever guidance is unavailable. Then*

$$\mathbb{E}\big[\#\{\textit{correct rows in } G^+(x)\} \mid \mathcal{F}\big] = \sum_{j=1}^{K'} q_j > \sum_{j=1}^{K'} p_j.$$

For an unguided fallback, set $Q_j = P_j = \pi_\theta(\cdot \mid x)$, so $q_j = p_j$.

Proof. Let $s_j = 1$ for a fresh global rollout and $s_j = k_j$ for a resumed rollout. For any correct completion $\sigma = (a_{s_j}, u_{s_j}, \ldots, a_n, u_n)$ with $P_j(\sigma) > 0$, we can factor each probability into action terms and environment terms:

$$\frac{Q_j(\sigma)}{P_j(\sigma)} = \prod_{s=s_j}^{n} \frac{q_{j,s}(a_s \mid S_s, g_j)\, E_j(u_s \mid S_s, a_s)}{p_{j,s}(a_s \mid S_s)\, E_j(u_s \mid S_s, a_s)} = \prod_{s=s_j}^{n} \frac{q_{j,s}(a_s \mid S_s, g_j)}{p_{j,s}(a_s \mid S_s)} \geq 1.$$

The environment factors E_j cancel by (A3). The inequality follows from (A1), and it is strict on at least one correct completion that has positive probability. Summing over all correct completions

with $P_j(\sigma) > 0$ gives

$$Q_j(\mathcal{C}_j) \geq \sum_{\substack{\sigma \in \mathcal{C}_j \\ P_j(\sigma) > 0}} Q_j(\sigma) > \sum_{\substack{\sigma \in \mathcal{C}_j \\ P_j(\sigma) > 0}} P_j(\sigma) = P_j(\mathcal{C}_j) > 0.$$

The last inequality uses (A2). The same calculation applies to a full trajectory and to a suffix after a retained prefix. It also applies to stacked guidance. The initial group has no correct rows. By linearity of expectation, the expected number of correct rows in $G^+(x)$ is therefore the sum over the appended rows. Each appended row with usable guidance adds a strict increase, and each unguided fallback row has the same success probability as the unaided draw it is compared with. This use of linearity does not require independence between rows. □

Scope. These sufficient conditions are strong. An insight can help on average even if it lowers the probability of some action on some correct path. A chosen restart turn also need not leave a correct continuation possible, as (A2) requires. For example, a returned turn of 0 or 1 is raised to 2, so the retry keeps the first turn and any irreversible error in it. The environment comparison in (A3) also requires replay to restore a state that matches the retained observations; keeping their text alone does not ensure this.

The proposition concerns sampling probabilities, so it cannot predict how many dead groups a given run will rescue. It does not rank the placements, and it says nothing about tasks where the unaided policy never succeeds. It also does not guarantee that the update has a nonzero gradient, or that the policy improves after training. A sampled success next to the retained failures gives the group reward variance, but masks, clipping, probability floors, and gradient cancellation can still affect the update. Stripping and its fallback act on that later update, after sampling has finished.

J Additional Related Work

Other GRPO modifications. Dr. GRPO (Liu et al., 2025) changes reward and length normalization, DAPO (Yu et al., 2025) also uses asymmetric clipping, and HINT (Wang et al., 2026b) uses abstract meta-hints. These approaches change different parts of the training process. Our main comparisons focus on where teacher guidance is placed and whether it stays in the inputs to the update.

Self-distillation and token-level credit. Extra information can guide the learning signal as well as the sampled trajectory. SDPO (Hübotter et al., 2026) trains a student to match the token predictions that a model makes when it sees feedback. Path-conditioned self-distillation (Li et al., 2026) assigns credit to individual tokens in single-turn reasoning. SDAR (Lu et al., 2026), ADRS (Zhang et al., 2026a), AgentOPSD (Wang et al., 2026c), and SSPO (Wu et al., 2026) use different ways to score tokens or steps. GRSD (Zheng et al., 2026) additionally trains reflection, while Skill-SD (Wang et al., 2026a) summarizes trajectories into skills. These methods differ in the information their teachers receive and how those teachers are updated. Most of these methods change the training signal. Guided retries instead change the interaction itself, and PATH-GRPO then removes the insight from the signal.

Self-training and agent RL. Other approaches improve a model by generating data and then learning from it. ReST (Gulcehre et al., 2023) alternates generation with offline training on filtered outputs. Distillation instead trains a student to match a teacher's predictions on fixed inputs or inputs generated by the student (Hinton et al., 2015; Agarwal et al., 2024). For interactive agents, LOOP (Chen et al., 2025), WebRL (Qi et al., 2025), and Search-R1 (Jin et al., 2025) combine reinforcement learning with different rewards and data-collection procedures. SWE-RL (Wei et al., 2025) trains a model to generate patches. SWEET-RL (Zhou et al., 2025) uses privileged information to learn how to assign credit to turns in collaborative tasks. ArCHer (Zhou et al., 2024) uses hierarchical actor-critic learning. These methods were trained in different settings, so our matched baselines (Section 4.1) are instead adapted and run on our own task pools. Appendix A separately compares our AppWorld results with published systems and explains the differences in base models, training data, and adaptation method.

K PROMPTS

For each turn-level retry in a dead group, the teacher makes two separate calls (Section 3.1). The selector chooses a restart turn k from the failed attempt and the verifier feedback. The writer then uses the reference solution to write an insight for that turn. We use the same selector and writer settings in both domains and at both model scales. On TravelPlanner, both calls also receive the candidate pool.

The selector prompts and the user messages below are *abridged*. Bold text marks instructions we highlight, and blue terms in double brackets are placeholders. Blue boxes are system prompts, and orange boxes are user messages. The prompts call the insight a hint and the reference solution the gold solution. The two writer boxes are excerpts of the writer's system prompts. For readability, we split them into paragraphs and bullets and bold two key instructions, without changing what they ask.

K.1 TURN SELECTION

To choose where the policy should try again, the selector receives the task, the failed trajectory, its exit reason, and the descriptions of the failed checks. It does not receive the reference solution (Table 5). We ask it to find the commitment that caused the failures, and to choose a turn after any useful exploration, so that the retry keeps that exploration. The policy will replay those earlier turns before regenerating the restart turn and the rest of the trajectory (Section 3.2).

The selector returns a turn number, a short reason, and a confidence level. The prompt calls the restart turn `DIVERGENT_TURN`, and the value 0 means that the selector found no single turn that caused the failure. In our runs, the implementation maps 0, 1, and non-integer answers to turn 2, caps larger answers at the last turn, and accepts every confidence level. So every retry keeps the first turn. The Random turn ablation instead draws the restart turn uniformly at random, and the Midpoint turn ablation uses the middle turn of the failed trajectory. A trajectory receives no turn-level insight if it has fewer than two turns or if its selector call fails. When guidance is unavailable, we replace the retry with a fresh unguided rollout, so the budget stays the same.

The two selector prompts follow one template: the same role, the same definition of a commitment, the same backward search from the failed checks, and the same output format. They differ only where the environments differ. In AppWorld, the agent commits through code actions, while in TravelPlanner it commits through search, city, and transport choices. The TravelPlanner prompt therefore adds two rules that exist only for plans: an error in assembling already-gathered data points to the submit turn, and a city choice counts only if that city cannot satisfy the failed checks.

Selector: system prompt (AppWorld, abridged)

```
You are a failure analyst reviewing a student AI agent's FAILED
attempt at a task. The agent interacted with apps through a Python
REPL over multiple numbered turns.

You are shown how the attempt ended and which grading checks it
FAILED, with descriptions. You do NOT have the correct solution. Use
only the failed-check descriptions and the trajectory itself.

Your task: identify the SINGLE turn where the agent made the
commitment that caused those failures. A commitment is an action or
decision that locked the agent onto a path where the failed checks
CANNOT be satisfied, no matter what it does afterward.

- Work backward from the failed checks. For each failure ask which
action, or which MISSING action, made this check impossible to pass,
then find the turn that performed or omitted that action.
```

- **Do not pick an early turn just for safety.** If valid, reusable exploration exists before the commitment, the commitment turn is AFTER that exploration. Think of it as: which turn should you cut so that everything BEFORE it is still on a viable path?
- **Conversely, do not default to the last turn.** If missing information is what fails the grading, the divergent turn is where the agent should have gathered that information but did something else.

Output EXACTLY these three lines, nothing else:
DIVERGENT_TURN: <integer turn number, or 0 if no single divergence point>
REASON: <one short line explaining what that turn committed to that broke the checks>
CONFIDENCE: <high|medium|low>

Selector: system prompt (TravelPlanner, abridged)

You are a failure analyst reviewing a student AI agent's FAILED travel-planning attempt. The agent used search tools over multiple numbered turns to gather trip information, then submitted a final JSON itinerary that FAILED specific constraint checks.

You are shown the constraint checks the itinerary FAILED and the candidate data pool the agent could have searched. **You do NOT have the correct answer.** Use only this evidence and the trajectory itself.

Your task: identify the SINGLE turn where the agent made the commitment that caused those failures. **A commitment is a search decision or city/transport choice that locked the agent onto a path where the failed constraints CANNOT be satisfied,** no matter how well it assembles the plan afterward.

- Work backward from the failed checks: which gathered datum (or missing datum) makes each check impossible to pass? Then find the turn that introduced or omitted that datum.
- **Distinguish a BAD COMMITMENT from a BAD ASSEMBLY.** If every needed datum is already gathered and the final write used a wrong row, repeated an item, or formatted it incorrectly, the divergent turn is the assembly/submit turn itself.
- **Do not pick an early turn just for safety,** and **do not default to the last turn.**
- **A city choice is divergent only if that city is PROVABLY unable to satisfy the failed checks** (check the pool for valid lodging, food, and attractions).

Output EXACTLY these three lines, nothing else:
DIVERGENT_TURN: <integer turn number, or 0 if no single divergence point>
REASON: <one short line explaining what that turn committed to that broke the checks>
CONFIDENCE: <high|medium|low>

Selector: user message (AppWorld, abridged)

How this attempt failed (for your diagnosis - do not repeat this back to the agent):

- Outcome: [[EXIT_REASON]].

```
- Passed [[N_PASS]] of [[N_TESTS]] of the task's grading checks.
- Grading checks it FAILED (use to locate the misconception; do NOT
repeat or name the target):
    - [[FAILED CHECK]]  ...

The agent's attempt had [[N_TURNS]] numbered turns. Here is the
transcript:

[[NUMBERED TRANSCRIPT]]

Now identify the first divergent turn. Output the three lines exactly
as instructed.
```

K.2 Insight Generation

Once a restart turn is chosen, the writer receives the failure evidence (the exit reason and the failed checks), the transcript with turn k marked, and the reference solution. From the reference, the writer learns the intended result. We ask it to turn this result into a precise selection or procedure rule for each wrong subgoal or slot. Each rule should tell the policy what to inspect or compare in its own observations so that it can identify the correct result. The guidance must be usable at turn k, so it must not mention the final grading result.

To keep the guidance procedural, the AppWorld prompt forbids three kinds of literals: names of APIs, functions, methods, fields, and arguments; answer values; and calls or filters that the policy could copy. The TravelPlanner prompt forbids literal venue names, transport cities, prices, and exact room-type strings. It still allows guidance about how transport is structured. As with the selector, both writer prompts follow one template: read the reference, give one line per wrong subgoal or slot, and state a rule for finding the right result without naming it. Only the domain vocabulary changes, namely what counts as a literal and, for TravelPlanner, what a valid transport plan looks like.

Writer: system prompt, AppWorld (excerpt)

```
You are ALSO shown the REFERENCE (GOLD) SOLUTION, so you KNOW the
correct API and value for each subgoal that is (about to be) wrong.
Use that knowledge ONLY to sharpen your hint.

At turn k, for each wrong subgoal, give one line:
- WHAT is wrong (which subgoal, which constraint/step it breaks), and
- a tight SELECTION / PROCEDURE RULE that uniquely determines the
right result from the agent's OWN REPL: the exact search or inspection
to run, and the exact filter/comparison/check to apply to its own
output, so only one option survives.

NEVER name the literal: no API/function/method name, no field/argument
name, no answer value, no copy-pasteable call or filter. Say HOW to
find it, not WHAT it is. If a rule does not narrow to one option, add
the qualifier/subset/ordering constraint the task states that
disambiguates.

Write in the SECOND PERSON, present/imperative ("re-list them and
match ...", "before you submit, re-read the task sentence and confirm
every qualifier was applied"). Never a third-person post-mortem, no
turn numbers, no "the agent". Keep it consumable at turn k (no mention
of the final grading).

Naming a gold API/field/value is the worst leak: the hint is stripped
at inference, so it teaches the policy nothing and only distorts
training.
```

Writer: system prompt, TravelPlanner (excerpt)

```
This agent executes a correct plan but cannot fix a vague nudge. You
are shown the gold plan and the searched pool, so you KNOW the correct
choice for each thing that is (about to be) wrong. Use that knowledge
ONLY to sharpen your hint.

At turn k, for each wrong slot, give one line:
- WHAT is wrong (which slot, which constraint it breaks), and
- a tight SELECTION RULE that uniquely lands on the right value from
the agent's OWN search: the exact search to run and the exact
filter/comparison, so only one option survives.

NEVER name the literal value: no restaurant/attraction/hotel name, no
transport city, no price, no exact room-type string. Say HOW to pick
it, not WHAT it is. If a rule does not narrow to one option, add the
budget/diversity/house-rule filter that disambiguates.

Keep the transport structure correct: one leg per transition, a closed
circle if required, consistent modes, and each leg's origin = the
previous destination. Keep it consumable at turn k (no mention of the
final grading).
```

Writer: user message (AppWorld, abridged)

```
How the whole attempt ended (context for diagnosing turn k - the hint
must still be consumable at turn k and must NOT mention the outcome or
that a check failed):

- Outcome: [[EXIT_REASON]].
- Passed [[N_PASS]] of [[N_TESTS]] of the task's grading checks.
- Grading checks it FAILED: [[FAILED CHECK]]  ...

The reference (GOLD) solution for this task (grounding for locating
where turn k diverged; do NOT name, quote or paraphrase any API,
field, value or step from it in your hint):

[[REFERENCE SOLUTION]]

Here is the full failed transcript. The agent will resume and
regenerate from turn [[K]] (marked below).

[[MARKED TRANSCRIPT]]

Write the hints for the agent as it is about to write turn [[K]]: one
short line per DISTINCT wrong assumption you found at turn [[K]], as
many as there genuinely are and no more. Often that is one. Every line
must name a MISCONCEPTION, never an API, field or value from the gold,
and be consumable at turn [[K]].
```

After the retry, we strip the insight (Section 3.3). A strip check then looks for remaining copies: the global insight in the prompt and the turn-level insight in the response, by exact text match or by matching a run of its tokens. It logs these matches but does not reject trajectories, and it does not detect paraphrases.