LEARNING FROM EVOLVING PROMPTS: ADAPTIVE PROMPT TEACHING FOR REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Prompt optimization and reinforcement learning (RL) improve language models through different channels: one edits the text in the model's context, while the other updates its weights. Recent methods combine the two by alternately updating the prompt and model weights to improve performance. However, the benefit of an optimized prompt may fade as RL updates the weights, and the prompt may eventually become redundant or even harmful. We propose Adaptive Prompt Teaching (**APT**), an RL framework that uses evolving prompts as temporary teachers during training and tests whether any prompt is worth keeping at deployment. **APT** first samples each training problem from its original, non-optimized input and resamples it under teacher prompts only when every sample fails and thus provides no learning signal. It then trains the model on the guided responses with the teacher part removed from the input, allowing the model without the optimized prompt to learn the same behavior. Teacher prompts are periodically rewritten based on the model's recent failures and retired once the model performs as well without them. Since some guidance may remain unabsorbed in the weights, a separate step searches for a prompt that still improves the model with its updated weights and uses it as the final deployment prompt for evaluation. On HoVer, WebShop, and SATBench with `Qwen3-8B` and `Gemma3-12B`, **APT** achieves the highest mean accuracy in all six settings, 1.3 to 4.1 points above the strongest baseline.

1 INTRODUCTION

Large language models (LLMs) can be improved by changing either their weights or their context. Reinforcement learning with verifiable rewards (RLVR) updates the weights (Shao et al., 2024; Xin et al., 2024), while prompt optimization rewrites the instructions in the context (Khattab et al., 2024; Yuksekgonul et al., 2024; Agrawal et al., 2026). Each has its own limit. RLVR learns only from sampled responses that provide a learning signal. Under group-relative objectives such as GRPO (Shao et al., 2024), this excludes any problem on which every sampled response fails. Prompt optimization cannot change the model itself, and its gain lasts only while the prompt stays in context.

These opposing limitations have led to methods that evolve prompts and weights together (Soylu et al., 2024; Tiwari et al., 2026; Zhang et al., 2026a). Their common aim is the final performance of the resulting system: the prompt is searched by reflection or evolutionary selection as one more component to optimize alongside the weights, and the two are deployed together at inference. Under this view, a good prompt is simply one under which the model scores higher, so prompt search and weight updates become two optimizers for a single objective.

Combined with RL, however, a prompt can play two roles: a *teacher* during training and an instruction at deployment. As a teacher, it leads the model to successful responses it rarely generates on its own, creating a learning signal where RL would otherwise have none. Once the weights absorb this behavior through context distillation (Snell et al., 2022), the model fails on different problems, so the teacher should evolve over training. By the end of training, the evolved prompt may be redundant, harmful, or still useful, depending on how much of its guidance the weights have absorbed. Its deployment role should therefore be tested against the bare model rather than assumed.

Joint methods like FST (Tiwari et al., 2026) do not draw this distinction: they optimize prompts and weights to maximize performance with the prompt in context. Training with the optimized prompt in context can increase the score without improving the *bare model*, i.e., the model given only the

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.

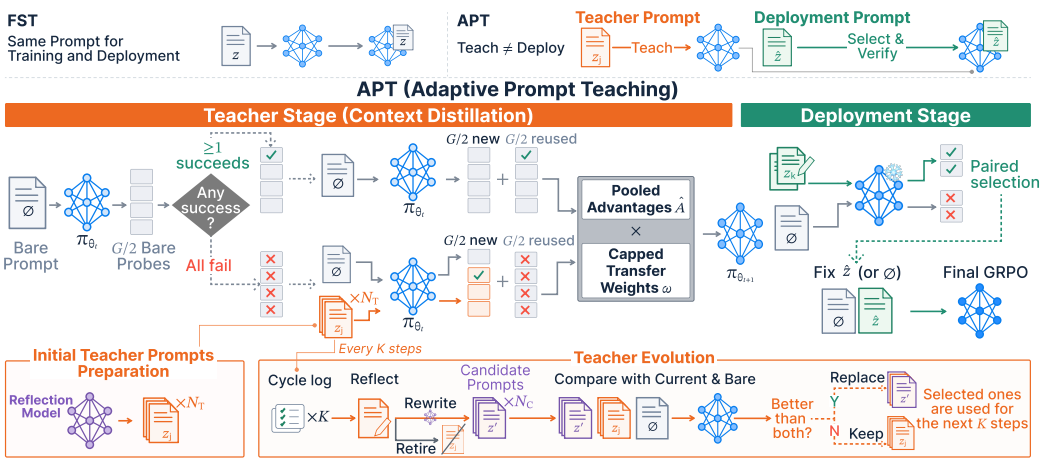


Figure 1: *Top:* Unlike joint-update methods, **APT** separates teacher and deployment prompts. *Bottom:* Overview of **APT**. Teachers guide only when all bare probes fail, and are distilled under the bare context. Teachers are rewritten, and retired once they stop beating bare sampling. A deployment prompt is kept only if a frozen-weight paired test favors it.

original input. Yet the value of updating weights through RL is that it makes useful behavior persist beyond the prompt that first elicited it. Keeping the prompt at deployment does not avoid this, since the gain may not hold if the prompt is paraphrased or removed. Training under a fixed prompt can also restrict exploration and limit how far GRPO improves the bare model. Some methods use prompts or hints as guidance only during training (Lu et al., 2026; Nath et al., 2025; Xia et al., 2026b), but these methods assume query-specific optimized guidance and do not account for the possibility that the model weights may not fully absorb the guidance. This motivates us to separate the prompt's role as a teacher during training from its role at deployment, deciding each relative to the bare model.

Contributions. We propose Adaptive Prompt Teaching (**APT**), an RL framework that uses an evolving prompt as a teacher and builds a separate deployment prompt when the model weights do not fully absorb the teacher guidance. During training, **APT** samples responses under teacher prompts only for problems where responses generated from the bare input fail, and learns from these guided responses with the teacher text removed, so that what the teacher elicits is distilled into the weights. Rather than relying on fixed guidance, **APT** reflects on whether the current teacher remains useful and rewrites it when necessary. A teacher is retired once the model performs as well without it. Finally, **APT** tests whether any prompt can still improve the trained model and deploys one only if it does. Our contributions are as follows:

1. **Context distillation from adaptive teacher guidance.** For problems the model fails to solve from the bare input, **APT** uses teacher prompts to elicit successful responses and trains on these responses with the teacher text removed, thereby distilling the teacher guidance into the model weights (§3.1, §3.3). Each teacher is rewritten by a reflection model based on the model's recent failures until a version proves useful against fresh bare-input responses, and is retired once the model performs as well without it. Thus, the guidance gradually fades as the model improves, all within GRPO's rollout budget. Additionally, we introduce two mechanisms to make this distillation more effective: a pooled advantage that credits guided responses against the whole group while keeping bare responses on their own baseline, and capped per-token transfer weights that prevent excessive drift when a guided response is too unlikely under the bare input (§3.2).

2. **Deployment prompt.** After teacher-guided training, **APT** builds deployment-prompt candidates from the training history, evaluates them against the bare model at frozen weights, and retains one only if it improves performance, using it for the remaining training updates (§3.4).

3. **Empirical gains and robustness.** On HoVer, WebShop, and SATBench with `Qwen3-8B` and `Gemma3-12B`, **APT** improves over GRPO by 6.5 points on average and outperforms the strongest baseline in each of the six settings by 1.3 to 4.1 points. We further show robustness to prompt choice and wording, and that using the same model as the reflector also works, suggesting potential for self-improvement.

2 Problem Formulation

We consider RLVR for an LLM policy π_θ over task instances $x \sim \mathcal{D}$ (the training set). The model input is built from x and a prompt z, a string of instruction text added to the context, and $\pi_\theta(y \mid x, z)$ is the induced distribution over responses $y = (y_1, \ldots, y_{|y|})$. We write $z = \varnothing$ for the *bare context*, the benchmark's original task interface with no added prompt, and call $\pi_\theta(\cdot \mid x, \varnothing)$ the *bare policy* (the bare model of §1) and its samples *bare responses*. A verifier assigns a reward $r(x, y) \in [0, 1]$, binary except on HoVer. The success rate of x under z and the population objective are

$$p_\theta(x; z) \;=\; \mathbb{E}_{y \sim \pi_\theta(\cdot|x,z)}\big[r(x, y)\big], \qquad J(\theta, z) \;=\; \mathbb{E}_{x \sim \mathcal{D}}\big[p_\theta(x; z)\big], \tag{1}$$

with $p_\theta(x) := p_\theta(x; \varnothing)$ denoting the bare success rate. At deployment, the system is characterized by both its model parameters θ and its prompt z, and its accuracy is $J(\theta, z)$.

GRPO. GRPO (Shao et al., 2024) samples G responses from $\pi_{\theta_{\text{old}}}(\cdot \mid x, z)$ and assigns each the advantage $\hat{A}_i = (r_i - \mu)/(\sigma + \eta)$, where μ and σ are the mean and standard deviation of the group's rewards and η is a small constant. The weights are updated with a clipped per-token surrogate plus a KL penalty to a reference policy with weight β. Two properties matter for what follows. **(P1)** A group whose rewards all agree has $\hat{A}_i \equiv 0$ and gives no policy-gradient signal, which under binary rewards happens with probability $p^G + (1-p)^G$ for $p = p_\theta(x; z)$, so problems the policy rarely solves teach almost nothing. **(P2)** If the responses are sampled under a nonempty prompt z, the gradient is taken under the same prompt, so the raw-reward update follows $\nabla_\theta J(\theta, z)$ in expectation and targets the prompted system (θ, z). The bare policy changes only because both contexts share the weights θ, and this change can raise or lower $J(\theta, \varnothing)$ (Proposition 1).

Prompt and weight optimization. Joint methods (Soylu et al., 2024; Tiwari et al., 2026; Zhang et al., 2026a) alternate prompt search with RL under the current prompt, which by (P2) improves the prompted policy, and deploy the resulting pair. Context distillation (Snell et al., 2022) instead samples responses under the prompt and takes the gradient under the bare context,

$$g_{\text{CD}}(x) \;=\; \mathbb{E}_{y \sim \pi_{\theta_{\text{old}}}(\cdot|x,z)}\Big[\hat{A}(x, y)\, \nabla_\theta \log \pi_\theta(y \mid x, \varnothing)\Big], \tag{2}$$

and methods that use guidance only during training build on it and deploy $(\theta, \varnothing)$ (Nath et al., 2025; Xia et al., 2026b; Lu et al., 2026). Because the responses come from the guided policy, Eq. (2) is in general a biased estimate of $\nabla_\theta J(\theta, \varnothing)$ (§3.2).

Teacher and deployment prompts. Both designs fix the deployment choice in advance, and we instead separate two roles. A *teacher prompt* z_j changes which responses are sampled during training but never the context of the gradient, so the updates it guides target the bare policy. A *deployment prompt* $\hat{z}$, possibly $\varnothing$, is chosen after this training and kept only if the *prompt-dependence gap*

$$\Delta(\theta, z) \;=\; J(\theta, z) - J(\theta, \varnothing) \tag{3}$$

is positive at the trained weights. We seek the deployed pair $(\theta, \hat{z})$ with the highest $J(\theta, \hat{z})$ within a fixed budget of training responses.

3 Method

Section 2 leaves two questions about a prompt that evolves during training. *How should it guide training?* Guidance is needed only where bare responses give nothing to learn from. What it elicits should be stored in the weights rather than tied to the prompt, and the prompt should be rewritten or retired as weights absorb it. *What should remain of it at deployment?* A prompt that helped training need not help the final policy, and one that still helps adds value the weights have not absorbed.

In this paper, we propose **APT**, which answers both questions. Throughout, a *teacher* is a prompt that guides the policy during training, not the reflection *model* that writes it. For each training problem, **APT** samples some responses under teachers only when bare sampling gives no learning signal (§3.1). It then learns from these guided responses under the bare context so that what a teacher elicits is stored in the weights (§3.2). As the weights absorb this behavior, a teacher that no longer improves on bare sampling is rewritten or retired (§3.3). Since absorption may still be incomplete after training with teachers, **APT** finally tests whether any prompt still improves the trained policy and keeps one

for deployment only if it does (§3.4). The *teacher stage* covers the first 80% of updates, and the *deployment stage* covers the last 20%. Figure 1 gives an overview, and Algorithm 1 (Appendix B) gives the full procedure.

3.1 Teacher Prompts and Routing

Teachers. **APT** keeps three teacher prompts $\mathcal{T} = \{z_1, z_2, z_3\}$ in complementary formats: a short procedure, one worked example, and a list of likely failures with their corrections. Before training, the reflection model writes each from a few training questions and a format guideline. Each teacher is appended to the benchmark's system prompt and contains no problem-specific answers or entities, so it applies across all problems (Appendix C.6).

Routing. By (P1), a GRPO group gives no learning signal when all of its responses receive the same reward. A teacher can help only when every response fails, so **APT** uses teachers only there (Lemma 3). Each time a problem appears in a batch, **APT** first samples four bare responses, called *probes*, and then fills the remaining four of its $G = 8$ responses in one of two ways: *(1)* if any probe succeeds, it samples four more bare responses and trains on the group as in GRPO, and *(2)* if all probes fail, the problem is *routed*, and it samples one response from each of the three teachers plus one more bare response, the *bare control*. Since routing happens only when the probes fail, comparing a teacher against them would overstate its benefit. We therefore use the control, a fresh bare draw on the same problem, as an unbiased reference for measuring each teacher (§3.3, Lemma 2). As the bare policy improves, fewer problems are routed and training returns to plain GRPO. We also implement asynchronous updates so that sampling in two stages adds no extra generation round (Appendix C.4).

3.2 Context Distillation from Guided Responses

To make the bare policy do on its own what it does with a teacher prompt, **APT** learns from teacher responses by context distillation (Eq. (2)). A teacher response is sampled under z_j, and the policy is then updated on that response under the bare context. Every gradient is thus a gradient of the bare policy, and a teacher changes how responses are sampled, not the objective. Joint methods (Soylu et al., 2024; Tiwari et al., 2026) instead update the policy under the evolving prompt, which improves the prompted policy (P2) and ties the gain to the prompt. Proposition 1 formalizes this contrast for an idealized update with exact importance weights (proof in Appendix D).

Proposition 1 (Teachers change the estimator, not the objective)**.** *The idealized teacher-stage update, with exact sequence-level importance weights and raw rewards, is an unbiased estimate of* $\nabla_\theta J(\theta, \varnothing)$ *whenever each teacher policy covers the bare policy's support, so teachers can change its variance but not its expectation. A raw-reward update under a prompt* z *instead has expectation* $\nabla_\theta J(\theta, z)$*, whose inner product with* $\nabla_\theta J(\theta, \varnothing)$ *can be negative, so a small step on it can lower* $J(\theta, \varnothing)$*.*

In the idealized update, rewriting a prompt changes the estimator but not the objective, whereas in joint methods it changes the objective. Two problems remain in turning this idealized update into a practical one. *First*, a group mean shared by bare and teacher responses would penalize bare failures merely because a teacher succeeded, so a teacher would change not only which responses are sampled but also how bare responses are credited. *Second*, guided responses contain tokens the bare policy would almost never produce, which would dominate an unweighted update, and the exact importance weight is unbounded. **APT** addresses these with a pooled advantage and capped transfer weights.

Pooled advantage. We call a teacher response *valid* if its output follows the task format. For problem x, let $\mathcal{B}_x$ be its bare responses and $\mathcal{V}_x$ the bare responses together with the valid teacher responses. Let μ_B and μ_V be the mean rewards over $\mathcal{B}_x$ and $\mathcal{V}_x$, and σ_V the reward standard deviation over $\mathcal{V}_x$. The advantage of response i is then

$$\hat{A}_i = \begin{cases} (r_i - \mu_B)/(\sigma_V + \eta), & i \in \mathcal{B}_x, \\ (r_i - \mu_V)/(\sigma_V + \eta), & i \in \mathcal{V}_x \setminus \mathcal{B}_x, \\ 0, & \text{otherwise (invalid teacher response),} \end{cases} \tag{4}$$

where η is the small constant of §2. Bare responses are centered on their own mean, so whether a bare response is rewarded or penalized does not depend on the teachers, and an unrouted group, which contains no teacher responses, is trained exactly as in GRPO. Teacher responses are centered on the pooled mean, so a teacher response earns positive credit only when it beats that mean and

negative credit when it falls below it. On a routed problem, a successful teacher response thus gives the bare policy a positive example to learn from, and this is the learning signal that teachers add.

Capped transfer weights. A guided response can contain tokens that appear only because the teacher was in context, such as "following the steps above." The bare policy assigns these tokens low probability, and since low-probability tokens receive the largest gradients, they would dominate an unweighted update and teach the bare policy to copy the teacher's wording. **APT** therefore weights each token by the ratio of its bare-context probability to its generation probability, capped at 1 since the ratio grows without bound on tokens the teacher made unlikely:

$$\omega_{i,t} = \mathrm{sg}\left[\min\left\{1,\ \frac{\pi_{\theta_{\mathrm{old}}}(y_{i,t} \mid x, \varnothing, y_{i,<t})}{\pi_{\theta_i}(y_{i,t} \mid x, c_i, y_{i,<t})}\right\}\right], \tag{5}$$

where sg is stop-gradient, and θ_i and c_i are the weights and context under which response i was generated. A fresh bare response has $\theta_i = \theta_{\mathrm{old}}$ and $c_i = \varnothing$, so $\omega \equiv 1$. Tokens caused by the teacher thus receive small weights, and in expectation ω equals the overlap between the guided and bare next-token distributions, so a teacher token never counts for more than it would in a bare sample (Appendix D). The token weight $\lambda_{i,t}$ is $\omega_{i,t}$ for valid responses and 0 for invalid teacher responses, except that a successful teacher response the bare policy already finds plausible receives full weight (Appendix C.5). Among the caps we tested, 1 gives the highest accuracy on all three benchmarks, and removing the cap lowers accuracy on each (Appendix E.3).

Objective. Let $\rho_{i,t}(\theta) = \pi_\theta(y_{i,t} \mid x, \varnothing, y_{i,<t}) / \pi_{\theta_{\mathrm{old}}}(y_{i,t} \mid x, \varnothing, y_{i,<t})$ be the per-token ratio under the bare context. **APT** minimizes

$$\mathcal{L}(\theta) = -\frac{1}{Z} \sum_{x \in \mathrm{batch}} \sum_{i=1}^{G} \sum_{t} \lambda_{i,t}\, f\big(\rho_{i,t}(\theta), \hat{A}_i\big), \tag{6}$$

where f is the per-token surrogate of each benchmark's GRPO baseline (CISPO (MiniMax et al., 2025) or clipped PPO (Schulman et al., 2017)), and Z counts all response tokens in the batch including zero-weight ones, so that down-weighting one response does not rescale the others. We also keep GRPO's reference-KL penalty, weighted per token and evaluated under the bare context (Appendix C.5). For a group with only bare responses, such as an unrouted one, Eq. (6) is the GRPO update, except that the four probes were sampled one update earlier and carry the capped weight.

3.3 Retiring and Rewriting Teachers

A teacher that helps the policy early in training may stop helping later, since the weights absorb what it elicits and the remaining failures may require different guidance. The initial teachers may also be poor from the start, since they are written from only a few training questions. **APT** therefore keeps revising its teachers throughout training. It keeps three *roles*, one per format of §3.1. Each role holds one *current teacher* at a time, and the reflection model proposes *candidates* to replace it. Training proceeds in cycles of $K = 6$ updates, and current teachers change only at cycle boundaries.

Lift. **APT** judges every teacher prompt, current or candidate, by its *lift*, the mean reward gain of its responses over the bare control on the routed problems where it was sampled in a cycle, with an invalid teacher response counted as reward 0 (formal definition with Proposition 2 in Appendix D). With binary rewards, lift is the number of *rescues*, problems the prompt solves and the bare control does not, minus *harms*, the reverse, divided by the number of those problems.

Evolution. Each role (teacher prompt) follows four rules, and cycles where its teacher is not sampled are skipped when counting consecutive cycles (Appendix C.6).

1. **Propose.** At each boundary, the reflection model writes a candidate for each role from its recent failures, except a *proven* one with positive lift for two consecutive cycles.
2. **Test.** For one cycle, the candidate is sampled on routed problems together with the current teacher and the bare control, so all three answer the same problems at the same weights.
3. **Accept.** The candidate replaces the current teacher only if its lift is positive and its mean reward exceeds the current teacher's on the problems both answered. Otherwise it is discarded.
4. **Retire.** A current teacher with lift ≤ 0 for two consecutive cycles is retired, and bare responses fill its slot in routed groups. The role still receives candidates while retired, and one with positive lift reactivates the role.

Table 1: Test accuracy (%), mean ± std over 3 training seeds (over three evaluation seeds for the untrained Base). Each method uses the prompt it deploys, which for **APT** is the selected deployment prompt, or the bare context if none qualifies. Avg. and the last row are unweighted means over benchmarks and over all six cells. Bold marks the best result per row.

			Baselines					Ours
Model	**Benchmark**	Base	GRPO	Seq	Alternate	FST	P^2O	**APT**
Qwen3-8B	HoVer	$2.03_{\pm 0.28}$	$48.9_{\pm 0.4}$	$49.1_{\pm 0.8}$	$49.5_{\pm 0.4}$	$48.4_{\pm 0.3}$	$49.4_{\pm 0.6}$	$\mathbf{53.6}_{\pm 0.7}$
	WebShop	$18.60_{\pm 0.71}$	$81.3_{\pm 2.1}$	$84.2_{\pm 1.6}$	$83.5_{\pm 1.0}$	$85.3_{\pm 0.2}$	$83.3_{\pm 0.3}$	$\mathbf{89.1}_{\pm 0.2}$
	SATBench	$65.00_{\pm 2.78}$	$70.2_{\pm 0.5}$	$71.7_{\pm 1.5}$	$70.8_{\pm 2.0}$	$71.2_{\pm 0.4}$	$69.0_{\pm 0.5}$	$\mathbf{73.0}_{\pm 1.6}$
	Avg.	$28.54_{\pm 0.79}$	$66.8_{\pm 0.9}$	$68.3_{\pm 1.2}$	$67.9_{\pm 0.6}$	$68.3_{\pm 0.2}$	$67.2_{\pm 0.4}$	$\mathbf{71.9}_{\pm 0.3}$
Gemma3-12B	HoVer	$0.56_{\pm 0.04}$	$39.0_{\pm 7.2}$	$52.3_{\pm 0.5}$	$50.9_{\pm 2.2}$	$50.3_{\pm 0.5}$	$44.2_{\pm 8.3}$	$\mathbf{53.8}_{\pm 0.2}$
	WebShop	$11.13_{\pm 0.19}$	$79.6_{\pm 2.9}$	$80.3_{\pm 1.1}$	$72.5_{\pm 2.0}$	$79.8_{\pm 0.3}$	$79.3_{\pm 0.8}$	$\mathbf{83.8}_{\pm 1.4}$
	SATBench	$51.90_{\pm 1.18}$	$53.9_{\pm 0.9}$	$53.0_{\pm 4.2}$	$54.3_{\pm 1.9}$	$56.0_{\pm 1.7}$	$56.7_{\pm 1.4}$	$\mathbf{58.4}_{\pm 0.9}$
	Avg.	$21.20_{\pm 0.44}$	$57.5_{\pm 3.3}$	$61.8_{\pm 0.9}$	$59.2_{\pm 1.0}$	$62.0_{\pm 0.5}$	$60.1_{\pm 2.9}$	$\mathbf{65.3}_{\pm 0.6}$
All (macro avg.)		$24.87_{\pm 0.61}$	$62.1_{\pm 1.7}$	$65.1_{\pm 1.1}$	$63.6_{\pm 0.2}$	$65.2_{\pm 0.2}$	$63.6_{\pm 1.5}$	$\mathbf{68.6}_{\pm 0.2}$

Since candidates fill teacher slots of routed groups and are trained on like other teacher responses (Table 7), teacher evolution adds no rollouts beyond GRPO and at most one reflection call per role per cycle. If a teacher's responses become indistinguishable from bare ones, its expected lift is at most zero, and for any $m \geq 1$, the chance that it survives $2m$ sampled cycles without becoming eligible for retirement is at most $(3/4)^m$ (Proposition 2). This makes guidance fade by measurement rather than by schedule, as a teacher is used only when all bare probes fail and while it beats a bare response.

3.4 Deployment Prompt Selection and Final Training

After the teacher stage, the weights may not have absorbed all behavior that a prompt can elicit, so a prompt can remain useful at deployment. At update $0.8N$, **APT** freezes the weights and runs a reflective prompt search in the spirit of GEPA (Agrawal et al., 2026). The reflection model proposes deployment candidates from each teacher role's history and the policy's recent failures, and the latest teacher texts are also tested as they are (Appendix C.7). Each candidate and the bare context answer the same next $M = 256$ training tasks, and a candidate qualifies only if its net rescues against the bare context are significantly positive under a paired bootstrap test, Bonferroni-corrected over candidates. This net gain estimates the prompt-dependence gap $\Delta(\theta, z)$ of Eq. (3), and **APT** selects the qualifying candidate with the largest bound as $\hat{z}$, or sets $\hat{z} = \varnothing$ if none qualifies.

The remaining 20% of updates are GRPO under the fixed pair $(\theta, \hat{z})$, which is plain GRPO when $\hat{z} = \varnothing$. Unlike a teacher, $\hat{z}$ is part of the deployed system, so by (P2) training under it trains exactly that system. Selection is the only step that adds rollouts beyond GRPO, at most $2{,}048$ episodes, much fewer than FST's prompt search (Appendix E.6).

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate on three benchmarks spanning single-turn reasoning and multi-turn interaction. **HoVer** (Jiang et al., 2020) asks for one follow-up Wikipedia query given a claim and a retrieval summary, scored by gold-title coverage of that query without executing retrieval (2,549 train / 791 test). **WebShop** (Yao et al., 2022) is a simulated store with a 15-turn budget and binary purchase success (5,410 / 500). **SATBench** (Wei et al., 2025) contains logical puzzles generated from SAT formulas, scored by the extracted final label (302 / 70). No examples are set aside as a separate prompt-selection split, and test data never reach a reflection model (Appendix C.1).

Models. Our primary policy is `Qwen3-8B` (Yang et al., 2025), run in thinking mode on HoVer and SATBench and without thinking on WebShop, whose environment requires short actions inside a long trajectory. We additionally report `Gemma3-12B` (Gemma Team et al., 2025), a text-only checkpoint with no thinking channel, to test transfer across model families. We use `Claude-Opus5` (Anthropic, 2026) for the reflection model that edits the prompts.

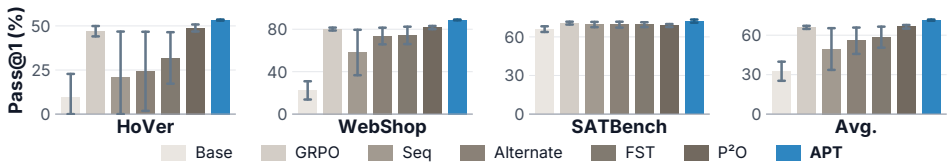


Figure 2: Prompt robustness on `Qwen3-8B`. The final checkpoint of each of three training seeds is evaluated under five contexts (Appendix C.8). Bars show accuracy averaged over seeds and then over contexts, with 95% t-intervals across contexts.

Baselines. All methods start from the same bare context $\varnothing$ and training stream. **GRPO** trains on eight bare responses per problem. **Seq** first optimizes a prompt at frozen weights (**Prompt-only**), then trains the model under it, counting both stages. **FST** (Tiwari et al., 2026) interleaves RL with GEPA (Agrawal et al., 2026) search over eight prompts and deploys its best prompt. **Alternate** is a single-candidate version of FST that periodically proposes one prompt and trains under whichever of it and the current prompt scores higher. **P^{2}O** (Lu et al., 2026) trains under problem-specific templates on hard problems and is evaluated bare. Appendix C.3 gives each procedure and its search budget.

Implementation Details. Within each model–benchmark cell every weight-training method shares the optimizer, decoding, data order, and horizon: 32 problems $\times$ $G = 8$ responses per update, 200 steps on HoVer and WebShop and 50 on SATBench, on eight GPUs per run (Appendix C.2). We report official binary correctness, averaged over responses within a task and then across tasks, at the final checkpoint, drawing four responses per task at temperature 0.6 and top-p 0.95 on HoVer and SATBench and one greedy trajectory on WebShop. Four draws therefore estimate expected single-response accuracy rather than pass@4. Every method is scored in the context it deploys, which is the bare context for GRPO and P^{2}O. For **`APT`**, this is $J(\theta, \hat{z})$, which reduces to $J(\theta, \varnothing)$ when no candidate qualifies, and we also report its bare score otherwise (Appendix E.2).

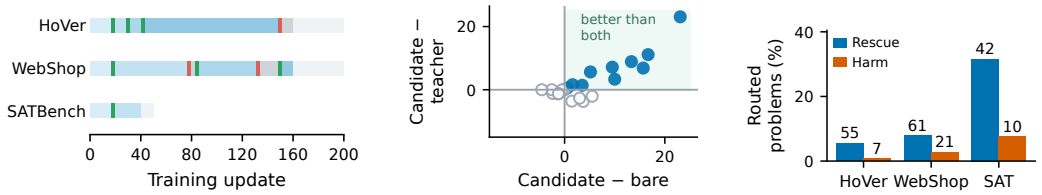
4.2 Main Results

Table 1 reports test accuracy for **`APT`** and the baselines. **`APT`** achieves the best result in every model–benchmark cell, outperforming GRPO by 6.5 points and FST, the strongest baseline on average, by 3.4 points averaged over all six cells. Its margin over the strongest baseline in each cell ranges from 1.3 to 4.1 points, although which baseline is strongest varies across cells among Seq, Alternate, FST, and P^{2}O. The gains are largest on WebShop, the multi-turn agentic benchmark, where **`APT`** leads the strongest baseline by 3.8 and 3.5 points on `Qwen3-8B` and `Gemma3-12B`, and on HoVer, where the margin reaches 4.1 points on `Qwen3-8B`.

The baselines also show where the gains come from. Seq and FST perform almost identically (65.1 and 65.2 on average), although FST searches a population of eight prompts throughout training and Seq optimizes a single prompt once before RL. When the prompt is optimized as one more component of the deployed system, extensive search therefore adds little. P^{2}O is closer to **`APT`** in that it learns from guided responses and deploys the bare model, yet **`APT`** exceeds it by 5.0 points on average. **`APT`** differs in three ways: its teachers are shared across problems and evolve with the policy, capped transfer weights limit what is copied from them, and a deployment prompt is kept when it still helps.

`APT` also reaches these results at a lower cost than the joint baselines. It uses teachers only on problems where every bare response fails, evaluates them on responses already generated for training, and calls the reflection model only for teachers that have not yet proven useful (Table 13, Appendix E.6). FST, in contrast, runs GEPA over a population of eight prompts throughout training. Together, these results indicate that a prompt is more valuable as a teacher whose guidance is distilled into the weights than as one more component of the deployed system.

Robustness to prompts. We re-evaluate each trained checkpoint at frozen weights under five system contexts: the bare prompt B_0, three hand-written variants of it, and the method's own optimized prompt, or Prompt-only's prompt for methods that deploy bare (Appendix C.8). As Figure 2 shows, methods that train under their optimized prompt, namely Seq, Alternate, and FST, perform well under that prompt but substantially degrade under the others. GRPO, P^{2}O, and **`APT`** update the weights under the bare context, so what they learn is stored in the weights rather than tied to a prompt, and their accuracy varies much less. Among all methods, **`APT`** achieves the highest mean accuracy.



(a) Prompt lifecycles. (b) Prompt-update selection. (c) Teacher rescues and harms.

Figure 3: **Teacher evolution** (`Qwen3-8B`). **(a)** Lifecycle of the worked-example role (S2). Each blue shade is one teacher version, green and red lines mark accepted rewrites and retirements, gray spans mark retired periods, and the pale span marks the deployment stage. **(b)** The HoVer candidates by mean reward gain ($\times 100$) over the bare control and over the current teacher (bare control only for a retired role). Filled points were accepted. **(c)** Rescues (blue) and harms (orange) of candidates against the bare control on routed problems.

(a) Bare context

SYSTEM: B0 (task + action format)
USER: current page + valid actions

(b) Teacher guidance

SYSTEM: B0 + teacher

"[…] NO-OPTION ITEMS […]: if the product page shows no color/size buttons, success depends only on type + attributes. […]

One click[description] then click[< prev] is allowed to check, then click[buy now]."

(c) Deployment guidance

SYSTEM: B0 + deployment

"[…] If it lists none — only descriptive phrases and a price cap — stop hunting option buttons: […]

If the title is silent about a phrase, click[description] once; […]"

Figure 4: Example contexts from one WebShop run of **APT** with `Qwen3-8B`: (a) the bare context, (b) a teacher prompt, and (c) the selected deployment prompt.

4.3 Analysis

Teachers turn over as the bare policy improves. Figure 3a tracks the second teacher role (S2) on `Qwen3-8B` for a single seed. On every benchmark, the initial teacher prompt is replaced by update 18. The HoVer role is retired at update 150, and the WebShop role is retired and reactivated twice. The appropriate guidance therefore changes as the policy weights evolve, and for some updates the best guidance is none. Figure 3b shows that on HoVer, the reflector proposes many candidate prompts, but fewer than half are accepted because a candidate must clearly outperform both the current teacher and the bare prompt. Figure 3c shows that candidates rescue many routed problems, so guidance creates learning signal where bare sampling gives none. Harms also occur, showing that routed problems are not always beyond the bare policy. This is why lift uses a fresh bare control rather than the probes, which were selected for failing (Lemma 2).

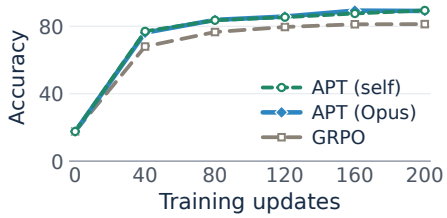
Teacher and deployment prompts can encode the same behavior in different words. Figure 4 shows one WebShop lineage of **APT** (`Qwen3-8B`, seed 0). The highlighted phrases mark a shared check: read the product description once before buying an item without options. The teacher embeds it in a full shopping procedure, whereas the deployment prompt (update 160) keeps only this check. This shows the behavior stays useful as explicit context, not that the deployment prompt derives from the teacher or that the weights cannot absorb it.

Ablation study. Figure 5a ablates the components of **APT** on `Qwen3-8B`. The full method reaches 71.9% average accuracy across the three benchmarks. Removing failure routing causes the largest drop (64.6%). Disabling teacher evolution or transfer weights lowers the average to 68.7% and 69.5%, respectively, while removing teacher retirement gives 71.0%. SATBench is unaffected because its teacher is rarely retired. Replacing the pooled advantage with a shared group mean gives 70.9%, and removing deployment selection similarly lowers the average to 70.8%. HoVer is unchanged in the latter since **APT** already deploys the bare prompt there, while WebShop and SATBench decline. Overall, these ablations support selectively using, evolving, and retiring teacher guidance while appropriately weighting the responses it elicits.

Potential for self-improvement. Our main experiments use `Claude-Opus5` as the reflection model. To test whether **APT** supports self-improvement, we instead let `Qwen3-8B` edit teacher prompts for a `Qwen3-8B` policy. Figure 5b compares this self-teaching variant with the Opus-edited

Variant	HoVer	WebShop	SATBench	Avg.
APT	**53.6**	**89.1**	**73.0**	**71.9**
w/o deploy. selection	**53.6**	88.0	70.7	70.8
w/o transfer weights	50.1	87.8	70.7	69.5
w/o pooled advantage	52.4	88.6	71.7	70.9
w/o retirement	52.6	87.8	72.7	71.0
w/o routing	48.2	86.8	58.9	64.6
w/o evolution	49.4	87.8	69.0	68.7

(a) Component ablations.



(b) Self-teaching versus GRPO.

Figure 5: Ablations and self-teaching on `Qwen3-8B`. **(a)** Component ablations of **APT**. Entries report mean accuracy (%) over three training seeds, and **Avg.** is the average over the three benchmarks. **(b)** WebShop test accuracy (%) over 200 updates for GRPO and for **APT** with `Claude-Opus5` or `Qwen3-8B` itself (self-teaching) as the reflection model.

version and GRPO on WebShop. Surprisingly, the self-teaching variant performs similarly to the Opus-edited version. At step 200, self-teaching reaches 89.4% accuracy, compared with 89.1% for the Opus-edited version and 81.3% for GRPO. This gives initial evidence that **APT** retains its gains without a stronger external editor. Additional results are provided in Appendix E.5.

5 Related Work

Learning signal in RLVR. RLVR improves reasoning from automatically verified outcomes (Shao et al., 2024; Xin et al., 2024; MiniMax et al., 2025), but group-relative objectives give no signal when all responses in a group have the same reward. To avoid this, one line of work changes curricula, rollout allocation, or subset selection (Foster et al., 2025; Li et al., 2026b; Xu et al., 2026), or reshapes advantages for zero-variance groups (Le et al., 2026). Another adds guidance such as partial solutions, answer information, or hints (Liu et al., 2025; Zhang et al., 2026d; Li et al., 2026a; Yan et al., 2025; Liao et al., 2026; Xia et al., 2026b), often only where unguided sampling fails. GUIDE (Nath et al., 2025), HiLL (Xia et al., 2026b), and P^2O (Lu et al., 2026) learn from guided responses under the unguided context, in the spirit of context distillation (Askell et al., 2021; Snell et al., 2022), but their guidance is written for individual problems and none of it is kept at deployment. In **APT**, teachers are shared natural-language prompts that are rewritten or retired once they stop beating a fresh bare response, capped per-token weights limit how much guided behavior is distilled, and a separate test decides whether any prompt remains at deployment.

Prompt and weight optimization. Prompt optimization revises instructions, demonstrations, or other context of a frozen model against a task metric, from DSPy (Khattab et al., 2024) and TextGrad (Yuksekgonul et al., 2024) to reflective search in GEPA (Agrawal et al., 2026), and later work optimizes playbooks, retrieved hints, episodic memories, harness code, and skills (Zhang et al., 2026b; Nekoei et al., 2025; Zhang et al., 2026c; Lee et al., 2026; Ye et al., 2026). Joint methods also update the weights, interleaving prompt optimization with fine-tuning (Soylu et al., 2024) or RL (Tiwari et al., 2026; Bo et al., 2026), co-evolving a prompt population with prompt-conditioned RL (Zhang et al., 2026a), or extending the idea to skill libraries (Xia et al., 2026a; Shi et al., 2026). Both deploy the optimized prompt and joint methods also train under it, so the prompt's role is fixed in advance. **APT** uses the same reflective revision, but its prompts serve as teachers only while they help the model, and a prompt remains at deployment only if a paired test at frozen weights favors it.

6 Conclusion

We presented **APT**, an RL framework that treats a prompt's role as something to measure against the bare model rather than fix in advance. During training, teacher prompts create learning signal on problems the bare policy cannot solve, their responses are distilled into the weights under the bare context, and each teacher is rewritten as the policy changes and retired once it stops beating bare sampling. After this teacher stage, a separate paired test decides whether any prompt still improves the trained model. Across three benchmarks and two model families, **APT** outperforms GRPO and the strongest baseline in every setting. We discuss limitations in Appendix A.

Reproducibility Statement

Section 3 describes **APT**, and Appendix B gives the full algorithm. Appendix C provides implementation details, including benchmarks, training settings, baselines, and prompt and deployment rules. Appendix D states all assumptions and gives complete proofs. All benchmarks are public, and anonymized source code and configuration files are included in the supplementary material.

LLM Usage

We used large language models to check grammar, polish the writing, and help implement parts of the code, such as the visualization scripts. The authors reviewed all such outputs and take full responsibility for the paper's content. Separately, `Claude-Opus5` serves as the reflection model within **APT** and the prompt-optimization baselines (§4).

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026.

Anthropic. System card: Claude opus 5. Technical report, Anthropic, July 2026. URL https://www-cdn.anthropic.com/c5fbac3f0b1280a933ebd26d3cb8bb9f5bdeaf48/Claude%20Opus%205%20System%20Card.pdf. Accessed: 2026-09-16.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

Xiaohe Bo, Rui Li, Zexu Sun, Quanyu Dai, Zeyu Zhang, Zihang Tian, Xu Chen, and Zhenhua Dong. Prompt and parameter co-optimization for large language models. In *International Conference on Learning Representations*, 2026.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on Machine Learning*, 2018.

Thomas Foster, Anya Sims, Johannes Forkel, Mattie Fellows, and Jakob Foerster. Lilo: Learning to reason at the frontier of learnability. In *Advances in Neural Information Processing Systems*, 2025.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Riviére, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt

Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report. *ArXiv*, abs/2503.19786, 2025. URL https://arxiv.org/abs/2503.19786.

Yichen Jiang, Shikha Bordia, Zheng Zhong, Charles Dognin, Maneesh Singh, and Mohit Bansal. HoVer: A dataset for many-hop fact extraction and claim verification. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, Online, 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.findings-emnlp.309/.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*, 2024.

Thanh-Long V Le, Myeongho Jeon, Kim Vu, Viet Lai, and Eunho Yang. No prompt left behind: Exploiting zero-variance prompts in llm reinforcement learning via entropy-guided advantage shaping. In *International Conference on Learning Representations*, 2026.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Jiazheng Li, Hongzhou Lin, Hong Lu, Kaiyue Wen, Zaiwen Yang, Jiaxuan Gao, Yi Wu, and Jingzhao Zhang. Questa: Expanding reasoning capacity in llms via question augmentation. In *International Conference on Learning Representations*, 2026a.

Ziniu Li, Congliang Chen, Tianyun Yang, Tian Ding, Ruoyu Sun, Ge Zhang, Wenhao Huang, and Zhi-Quan Luo. Knapsack rl: Compute-efficient reinforcement learning via heterogeneous rollout allocation. In *International conference on Machine Learning*, 2026b.

Baohao Liao, Hanze Dong, Xinxing Xu, Christof Monz, and Jiang Bian. Self-hinting language models enhance reinforcement learning. *arXiv preprint arXiv:2602.03143*, 2026.

Ziru Liu, Cheng Gong, Xinyu Fu, Yaofang Liu, Ran Chen, Shoubo Hu, Suiyun Zhang, Rui Liu, Qingfu Zhang, and Dandan Tu. Ghpo: Adaptive guidance for stable and efficient llm reinforcement learning. *arXiv preprint arXiv:2507.10628*, 2025.

Xinyu Lu, Kaiqi Zhang, Jinglin Yang, Boxi Cao, Yaojie Lu, Hongyu Lin, Min He, Xianpei Han, and Le Sun. P^2O: Joint policy and prompt optimization. *arXiv preprint arXiv:2603.21877*, 2026.

MiniMax, Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, Chengjun Xiao, Chengyu Du, Chi Zhang, Chu Qiao, Chunhao Zhang, Chunhui Du, Congchao Guo, Da Chen, Deming Ding, Dianjun Sun, Dong Li, Enwei Jiao, Haigang Zhou, Haimo Zhang, Han Ding, Haohai Sun, Haoyu Feng, Huaiguang Cai, Haichao Zhu, Jian Sun, Jiaqi Zhuang, Jiaren Cai, Jiayuan Song, Jin Zhu, Jingyang Li, Jinhao Tian, Jinli Liu, Junhao Xu, Junjie Yan, Junteng Liu, Junxian He, Kaiyi Feng, Ke Yang, Kecheng Xiao, Le Han, Leyang Wang, Lianfei Yu, Liheng Feng, Lin Li, Lin Zheng, Linge Du, Lingyu Yang,

Lunbin Zeng, Minghui Yu, Mingliang Tao, Mingyuan Chi, Mozhi Zhang, Mujie Lin, Nan Hu, Nongyu Di, Peng Gao, Pengfei Li, Pengyu Zhao, Qibing Ren, Qidi Xu, Qile Li, Qin Wang, Rong Tian, Ruitao Leng, Shaoxiang Chen, Shaoyu Chen, Shengmin Shi, Shitong Weng, Shuchang Guan, Shuqi Yu, Sichen Li, Songquan Zhu, Tengfei Li, Tianchi Cai, Tianrun Liang, Weiyu Cheng, Weize Kong, Wenkai Li, Xiancai Chen, Xiangjun Song, Xiao Luo, Xiao Su, Xiaobo Li, Xiaodong Han, Xinzhu Hou, Xuan Lu, Xun Zou, Xuyang Shen, Yan Gong, Yan Ma, Yang Wang, Yiqi Shi, Yiran Zhong, Yonghong Duan, Yongxiang Fu, Yongyi Hu, Yu Gao, Yuanxiang Fan, Yufeng Yang, Yuhao Li, Yulin Hu, Yunan Huang, Yunji Li, Yunzhi Xu, Yuxin Mao, Yuxuan Shi, Yuze Wenren, Zehan Li, Zelin Li, Zhanxu Tian, Zhengmao Zhu, Zhenhua Fan, Zhenzhen Wu, Zhichao Xu, Zhihang Yu, Zhiheng Lyu, Zhuo Jiang, Zibo Gao, Zijia Wu, Zijian Song, and Zijun Sun. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *ArXiv*, abs/2506.13585, 2025. URL https://arxiv.org/abs/2506.13585.

Vaskar Nath, Elaine Lau, Anisha Gunjal, Manasi Sharma, Nikhil Baharte, and Sean Hendryx. Adaptive guidance accelerates reinforcement learning of reasoning models. *arXiv preprint arXiv:2506.13923*, 2025.

Hadi Nekoei, Aman Jaiswal, Patrice Bechard, Oleh Shliazhko, Orlando Marquez Ayala, Mathieu Reymond, Massimo Caccia, Alexandre Drouin, Sarath Chandar, and Alexandre Lacoste. Jefhinter: Leveraging offline knowledge for improving web agents adaptation. *arXiv preprint arXiv:2510.04373*, 2025.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, HaoWei Zhang, Mingchuan Zhang, Y.-K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv*, abs/2402.03300, 2024. URL https://api.semanticscholar.org/CorpusID:267412607.

Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi Gu, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*, 2026.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022.

Dilara Soylu, Christopher Potts, and Omar Khattab. Fine-tuning and prompt optimization: Two great steps that work better together. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.emnlp-main.597/.

Rishabh Tiwari, Kusha Sareen, Lakshya A Agrawal, Joseph E Gonzalez, Matei Zaharia, Kurt Keutzer, Inderjit S Dhillon, Rishabh Agarwal, and Devvrit Khatri. Learning, fast and slow: Towards llms that adapt continually. *arXiv preprint arXiv:2605.12484*, 2026.

Anjiang Wei, Yuheng Wu, Yingjia Wan, Tarun Suresh, Huanmi Tan, Zhanke Zhou, Sanmi Koyejo, Ke Wang, and Alex Aiken. Satbench: Benchmarking llms' logical reasoning via automated puzzle generation from sat formulas. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026a.

Yu Xia, Canwen Xu, Zhewei Yao, Julian McAuley, and Yuxiong He. Learning to hint for reinforcement learning. *arXiv preprint arXiv:2604.00698*, 2026b.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*, 2024.

Yixuan Even Xu, Yash Savani, Fei Fang, and J Zico Kolter. Not all rollouts are useful: Down-sampling rollouts in llm reinforcement learning. *Transactions on Machine Learning Research*, 2026.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. In *Advances in Neural Information Processing Systems*, 2025.

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

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 2022.

Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. Meta context engineering via agentic skill evolution. *arXiv preprint arXiv:2601.21557*, 2026.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation"' via text. *arXiv preprint arXiv:2406.07496*, 2024.

Lunjun Zhang, Ryan Chen, and Bradly C Stadie. Evolutionary system prompt learning for reinforcement learning in llms. *arXiv preprint arXiv:2602.14697*, 2026a.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026b.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, Bo Tang, and Muning Wen. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026c.

Xichen Zhang, Sitong Wu, Yinghao Zhu, Haoru Tan, Shaozuo Yu, Ziyi He, and Jiaya Jia. Scaf-grpo: Scaffolded group relative policy optimization for enhancing llm reasoning. In *International Conference on Learning Representations*, 2026d.

Algorithm 1: Adaptive Prompt Teaching (**APT**)

Require: Policy θ_0, training stream, reflection model, budget N, cycle length K

Write the initial teachers $\mathcal{T} = \{z_1, z_2, z_3\}$ with the reflection model from a few training questions, and generate the four bare probes of the first batch
for update $u = 1, \ldots, 0.8N$ **do**
  Draw a batch and route each problem whose four cached probes all failed (§3.1)
  Sample the second half of each group: four bare responses if unrouted, otherwise the bare control and one response per active teacher or candidate under test, bare in any empty slot
  Generate the four probes of the next batch concurrently
  Re-score all responses under $\varnothing$ and compute $\hat{A}_i$ and $\omega_{i,t}$ (Eqs. (4) and (5))
  Update θ by Eq. (6) and record the lift of each teacher and candidate (Eq. (14))
  if $u \bmod K = 0$ **then**
    Accept each candidate whose test cycle has ended if its lift is positive and it beats the current teacher, and discard it otherwise (§3.3)
    Retire at most one teacher whose lift has been nonpositive for two consecutive cycles
    Request a rewrite for every role that is not proven, and test each candidate whose reply has arrived during the next cycle
  end if
end for
Freeze θ and form the candidates $\mathcal{C}$, $|\mathcal{C}| \leq 7$: the latest text of each teacher role and four prompts written from the training history (§3.4)
Select $\hat{z}$ by paired bootstrap on $M = 256$ training tasks (Eq. (11)), otherwise $\hat{z} \leftarrow \varnothing$
for update $u = 0.8N + 1, \ldots, N$ **do**
  GRPO with G responses sampled and trained under $\pi_\theta(\cdot \mid x, \hat{z})$
end for
Evaluate $(\theta_N, \varnothing)$ and $(\theta_N, \hat{z})$ on test

A Limitations

Our experiments cover three benchmarks with verifiable rewards and two models with 8B and 12B parameters, and we have not tested **APT** with larger models, longer training horizons, or learned reward models. **APT** relies on a reflection model to generate teachers and deployment candidates. Using the policy itself retains gains over GRPO, but this may not hold for much smaller models, such as 1B models. Deployment selection may also be noisy because it relies on 256 training tasks to evaluate the effectiveness of candidate prompts. Regarding the propositions, the capped transfer weights make the teacher-stage update a biased surrogate for the bare policy gradient, and our propositions concern an idealized estimator rather than convergence of the implemented one.

B Algorithm

Algorithm 1 summarizes the full procedure of **APT**. The first loop is the teacher stage, which routes each problem by its bare probes, learns from guided responses under the bare context, and retires or rewrites teachers at cycle boundaries (§3.1–§3.3). The remaining steps form the deployment stage, which selects a deployment prompt from the training history at frozen weights and trains the final updates under it (§3.4).

C Implementation Details

This appendix gives the settings needed to reproduce Section 4, and the code is attached. It covers the data (§C.1), the shared training configuration (§C.2), the baselines (§C.3), the components of **APT** (§C.4–§C.7), and the robustness protocol (§C.8). All prompts are given in Appendix F.

Table 2: Benchmark interfaces and evaluation. Training reward and reported correctness differ only on HoVer, whose training reward is dense. WebShop allows 64 tokens per action.

	HoVer	WebShop	SATBench
Train / test tasks	2,549 / 791	5,410 / 500	302 / 70
Interaction	single turn	≤ 15 turns	single turn
Thinking (`Qwen3-8B`)	on	off	on
Training reward	title coverage	binary success	binary label
Reported correctness	full coverage	binary success	binary label
Max prompt / response tokens	4,096 / 8,192	20,480 / 20,480	8,192 / 16,384
Max context tokens	12,288	24,576	24,576
Test responses per task	4	1	4
Test decoding (τ / top-p)	0.6 / 0.95	greedy	0.6 / 0.95

Table 3: Settings shared by every method, model, and benchmark.

Setting	Value
Optimizer	AdamW
Learning rate	10^{-6}
Weight decay	0
Gradient-norm clip	1.0
Schedule	constant after warmup
Problems $\times$ responses per update	32×8
Optimizer updates per batch	1
Training temperature	1.0
Rollout engine	vLLM with prefix caching
GPUs per run	8
Reflection model	`Claude-Opus5`

C.1 Benchmarks and Data

Training stream. Each training split is shuffled once under the run seed and consumed in drop-last batches of 32 problems, so every method sees the same problems in the same order. No subset is held out: the problems used to compare or select prompts come from this stream and are also trained on, and the test split is never used to propose, compare, or select prompts.

Grading. HoVer scores the fraction of gold article titles found in the first line of the final fenced block, up to 32 words. Training uses this coverage as the reward, the reported metric is full coverage, and the adapter does not execute retrieval. WebShop uses the environment's binary full-success reward, and SATBench grades only the final `[SAT]`/`[UNSAT]` label.

C.2 Training Configuration

Table 3 lists the settings every method shares, Table 4 the per-benchmark constants, and Table 5 the only model-specific differences.

C.3 Baselines

Prompt-only, the search stage of Seq, and Alternate share one prompt-search schedule. After the first six updates and every six updates thereafter, the reflection model proposes a prompt from recent training traces, normally six failures and two successes (Appendix F.6). The candidate and the current prompt each answer the next 192 training problems once, and the candidate is kept only if its paired mean reward is strictly higher. These problems are then used by the method's own training stage. Seq runs the search to completion, resets to the original weights, and trains the full horizon under the selected prompt. FST interleaves RL with GEPA search over a population of eight prompts and deploys its best prompt with the trained weights. P^{2}O follows its published procedure with its native zero-KL objective. Table 6 compares the search budgets. All baselines replace the system prompt with their own, whereas **`APT`** keeps B_0 and appends to it.

Table 4: Per-benchmark settings. Baselines share the upper block, and the lower block is specific to **APT**. Text caps are in student tokens, and the promotion limit is a multiple of the batch median bare response (§C.5).

	HoVer	WebShop	SATBench
Total weight updates N	200	200	50
Warmup updates	10	0	10
Surrogate f	CISPO	clipped PPO	CISPO
Within-update ratio clip	$[0, 4]$	$[0.8, 1.28]$	$[0, 4]$
Negative-advantage dual clip	—	3	—
Reward std. normalization	yes	no	yes
KL coefficient β	10^{-3}	0	10^{-3}
Test interval (updates)	20	8	10
Teacher / deployment updates	160 / 40	160 / 40	40 / 10
Cycle length K (updates)	6	6	6
Teacher / reflected deployment text cap	1,024 / 128	1,024 / 128	1,024 / 128
Promotion length limit	$1.5\times$	$2\times$	none
Deployment selection tasks n	256	256	256
Checkpoint interval (updates)	20	20	10

Table 5: Model-specific settings; everything else is shared. The Gemma checkpoint is the text-only export (`Gemma3ForCausalLM`) of `google/gemma-3-12b-it`. *24,576 on SATBench.

	`Qwen3-8B`	`Gemma-3-12B-it`
Checkpoint	stock	text-only
Thinking	per benchmark	off
System message	system turn	start of first user turn
Parameter / optimizer offload	no	yes
vLLM memory fraction	0.60	0.45
Actor tokens per GPU	32,768*	max context length

C.4 **APT**: Teachers, Routing, and Sampling

Teachers. Before training, the reflection model writes each role's initial text from a few training questions and a guideline for its format (Appendix F.2), and all three are used from the first update. A teacher may explain the task interface but may not contain answers, entities, or problem-specific solutions, and hidden answers never enter the reflection model's evidence. Let B_0 be the benchmark's system prompt, so the bare context $\varnothing$ is B_0 alone. Teacher and deployment texts are appended to B_0 in the system message, and the user message is unchanged. Gemma 3 has no system turn, so its chat template places the system message at the start of the first user turn. For learning, every teacher response is re-rendered under the bare context: the teacher text is removed from every prompt segment, including later WebShop turns, and the response tokens are kept as generated.

Group composition. Table 7 gives the eight responses of each group. Slots 0–3 hold the probes, and slot 4 of a routed group holds the bare control, the only bare response used as a reference for lift. A retired role's slot is filled by a bare response, so every group has eight responses, at least five of them bare. In a test cycle (every K steps), a seeded balanced draw assigns each routed problem to one candidate (Appendix C.6).

Pipelined sampling. The probes of batch $b{+}1$ are generated together with the last four responses of batch b, so routing adds no generation round. Each problem still receives exactly eight generated and trained responses. The cached probes are one update old and receive the capped weight of Eq. (5).

C.5 **APT**: Credit and Loss

Advantages. In Eq. (4), cached and fresh bare responses share μ_B, and invalid teacher responses are left out of $\mathcal{V}_x$. HoVer and SATBench use $\eta = 10^{-6}$, and WebShop, like its GRPO baseline, uses

Table 6: Prompt-search cost per six-update cycle. Calls are reflection-model calls. Extra episodes are rollouts beyond the 1,536 training responses of a cycle. **APT** compares teachers on responses already generated for training, and its only extra rollouts are the one-time deployment search: four reflection calls and at most 2,048 episodes.

Method	Prompts	Calls	Extra episodes	Weights
Prompt-only	1	1	384	frozen
Seq (search / train)	1 / 1	1 / 0	384 / 0	frozen / trained
Alternate	1	1	384	trained
P^2O	population	as published	as published	trained
FST	8	as published	as published	trained
APT	3 teachers	≤ 3	0	trained

Table 7: Responses per problem. Routing depends only on slots 0–3, and slot 4 of a routed problem is a fresh bare draw. A retired role's slot samples bare, and in a retired role's test cycle the current-teacher slot goes to a second active teacher.

Problem	Slots 0–3	Slot 4	Slots 5–7
Unrouted	probes	bare	bare
Routed	probes	bare control	three teachers
Routed, test cycle	probes	bare control	candidate, current, one other
Last 20% of updates		eight responses under $B_0 + \hat{z}$	

centered rewards without division by $\sigma_V + \eta$. If all rewards in $\mathcal{V}_x$ agree, every advantage is zero. With group-normalized advantages and capped per-token weights, the practical update is in general a biased estimate of the bare-policy gradient in Proposition 1.

Token weights. In Eq. (5), the numerator is the bare-context probability at θ_{old}, and the denominator is the recorded generation probability: under z_j at θ_{old} for a teacher response, and under $\varnothing$ at the previous update's parameters for a cached probe. The token weight $\lambda_{i,t}$ is 1 for fresh bare responses, $\omega_{i,t}$ for cached probes and valid teacher responses, and 0 for invalid teacher responses. A valid teacher response is *promoted* to $\lambda_{i,t} = 1$ if its advantage is positive, its mean ω is at least 0.5, and its length is within the limit of Table 4, relative to the median bare response in the batch. Promotion does not change the KL weight, and it is the only case in which a teacher token can receive more weight than Lemma 1 allows.

Surrogates. Let $\ell_{i,t}(\theta) = \log \pi_\theta(y_{i,t} \mid x, \varnothing, y_{i,<t})$, so $\rho_{i,t}(\theta) = \exp[\ell_{i,t}(\theta) - \ell_{i,t}(\theta_{\text{old}})]$, and let $m_{i,t} \in \{0, 1\}$ mark nonpadding response tokens. The surrogate f of Eq. (6) is that of each benchmark's GRPO baseline. HoVer and SATBench use CISPO (MiniMax et al., 2025):

$$\mathcal{L}^{\text{CISPO}}(\theta) = -\frac{1}{Z} \sum_x \sum_i \sum_t m_{i,t}\, \lambda_{i,t}\, \text{sg}\Big[\hat{A}_i \,\text{clip}\big(\rho_{i,t}(\theta), 0, 4\big)\Big]\, \ell_{i,t}(\theta). \tag{7}$$

WebShop uses clipped PPO with dual clipping of negative advantages:

$$\begin{aligned} \psi_{i,t}(\theta) &= \min\Big\{\rho_{i,t}(\theta)\hat{A}_i,\ \text{clip}\big(\rho_{i,t}(\theta), 0.8, 1.28\big)\hat{A}_i\Big\}, \\ \psi^{\text{dual}}_{i,t}(\theta) &= \begin{cases} \psi_{i,t}(\theta), & \hat{A}_i \geq 0, \\ \max\Big\{\psi_{i,t}(\theta),\ 3\hat{A}_i\Big\}, & \hat{A}_i < 0, \end{cases} \\ \mathcal{L}^{\text{PPO}}(\theta) &= -\frac{1}{Z} \sum_x \sum_i \sum_t m_{i,t}\, \lambda_{i,t}\, \psi^{\text{dual}}_{i,t}(\theta). \end{aligned} \tag{8}$$

In both, $Z = \sum_x \sum_i \sum_t m_{i,t}$ counts every response token in the batch, including zero-weight ones. The reference-KL penalty is

$$\frac{\beta}{Z} \sum_x \sum_i \sum_t m_{i,t}\, \kappa_{i,t}\, d^{\text{KL}}_{i,t}(\theta), \tag{9}$$

Table 8: Reflection-model contracts. Malformed or oversized replies are rejected, and a truncated reply is never used.

	Teacher rewrite	Deployment proposal
When	every boundary, except a proven role	once, at $0.8N$
Calls	at most one pending per role	four, one per framing
Evidence	≤ 6 failures and ≤ 2 successes under the current text; the other two teachers	shared evidence packet (§C.7)
Output	`teacher_prompt`, `deployment_notes`, `portable_behavior`, `rationale`	`deployment_prompt`, `rationale`
Text cap	1,024 student tokens	128 student tokens
Input cap	24,000 student tokens	
Output budget	8,192 tokens with thinking; one 16,384-token retry on truncation	

where β is the GRPO baseline's coefficient, $d_{i,t}^{\mathrm{KL}}$ is the per-token KL estimator under the bare context, and $\kappa_{i,t}$ is 1 on bare tokens, $\omega_{i,t}$ on valid teacher tokens, and 0 on invalid ones. No gradient flows through rewards, advantages, or weights.

C.6 APT: Teacher Evolution

Table 8 gives the reflection model and its budgets, and Appendix F.3 gives the rewrite prompt.

Boundary order. At each cycle boundary, **APT** first checks the candidates whose test cycle just ended, then updates each role's lift streaks. Then it retires at most one role, and finally requests new rewrites if needed.

Rewrite requests. Every role that is not proven and has evidence from the last cycle receives a request, and a role has at most one pending request. The request contains the role's current text, the other two teachers' texts, and up to eight routed problems of the last cycle answered under the current text: up to six where the teacher failed, prioritizing low bare success, and up to two where it succeeded. Each problem carries the task, the teacher and bare responses, their rewards, and grader feedback. A retired role is no longer sampled, so its request carries recent bare failures instead. The reply is a complete replacement within the teacher length cap, together with short deployment notes that are archived for the deployment search (§C.7).

Timing. Rewrites run concurrently with training, and replies are collected only at the first update of a cycle. Thus **APT** is training the models even when reflection models are working to rewrite the teacher prompts. A candidate requested at boundary u is tested over updates $u{+}1$ to $u{+}6$ if its reply is ready at $u{+}1$. Otherwise it is tested over $u{+}7$ to $u{+}12$, and the run waits for the reply at $u{+}7$. A candidate is tested only if its test cycle ends within the teacher stage.

Test and acceptance. In the test cycle, each routed problem receives its assigned candidate z', the current teacher z_j of the same role, and one other active teacher, alongside the bare control y_x^{ctrl} (Table 7). Let $\mathcal{R}_{z'}$ be the routed problems assigned to z', and let $\tilde{r}$ be the reward of a valid response and 0 otherwise. z' is accepted if

$$\frac{1}{|\mathcal{R}_{z'}|}\sum_{x\in\mathcal{R}_{z'}}\left[\tilde{r}(x,y_x^{z'})-r(x,y_x^{\mathrm{ctrl}})\right]>\epsilon \quad \text{and} \quad \frac{1}{|\mathcal{R}_{z'}|}\sum_{x\in\mathcal{R}_{z'}}\left[\tilde{r}(x,y_x^{z'})-\tilde{r}(x,y_x^{z_j})\right]>\epsilon, \quad (10)$$

where the first term is $\mathrm{lift}(z')$ of Eq. (14), and $\epsilon = 10^{-9}$ keeps rounding error in dense HoVer rewards from counting as a gain. Lift uses the bare control rather than the probes, which were selected for failing. For a retired role, the other slots go to the active teachers, and only the first condition applies. A candidate with no routed problems is not evaluated. These lifts come from small samples, so they are noisy selection signals rather than statistical tests.

Table 9: Deployment candidates at $0.8N$. The last four are written by the reflection model, one per framing. Caps are in student tokens. The bare context $\varnothing$ is the fallback, not a candidate, so $|C| \leq 7$.

Candidate	Content	Cap
S1, S2, S3	latest teacher texts, used literally	1,024
Direct teacher reuse	one useful teaching instruction, minimally adapted	128
Residual procedure	a compact procedure for behavior the bare policy still misses	128
Worked demonstration	one or two small checked examples	128
Practical retained context	conditional checks, recovery, or stopping advice	128

Retirement. A lift of at most ϵ counts as nonpositive. A cycle in which the current teacher is not sampled leaves its streaks unchanged, and an accepted candidate's test cycle counts as its first cycle. When several roles have two consecutive nonpositive cycles, only the one with the lowest lift is retired at that boundary, with ties broken by the role order.

C.7 **APT**: Deployment Selection and Final Training

Candidates. At update $0.8N$, **APT** stops the pipeline, the policy and prompts are checkpointed, and the weights are frozen. The candidates are the latest text of each teacher role, active or retired, used literally, and four texts written by the reflection model, one per framing in Table 9, each at most 128 tokens (Appendix F.4). This is because longer and more specific prompts are redundant for a model whose weights have already absorbed the guidance. The four requests share one evidence packet: each teacher's per-cycle rescues, harms, and lift, the retirements and version history, the archived deployment notes, recent routed problems pairing a teacher response with the bare control, and recent bare responses. Each request also states the selection rule and lists the literal teachers as competitors. Duplicates are merged, so the tested set $\mathcal{C}$ has at most 7 candidates.

Selection. The bare context and every candidate answer each of the next $M = 256$ distinct training tasks once, with the test decoding of Table 2. With official correctness $s(x, y) \in \{0, 1\}$, which on HoVer is full title coverage, a candidate's paired gain is

$$\delta(z) = \frac{1}{M} \sum_{x} \left[s(x, y_x^z) - s(x, y_x^{\varnothing}) \right], \tag{11}$$

its rescues minus harms per task. A candidate qualifies if the one-sided lower bound of δ at level $1 - 0.05/|\mathcal{C}|$, from 10,000 paired bootstrap resamples of the tasks, is strictly positive. **APT** selects the qualifying candidate with the largest bound, then the largest δ, and otherwise sets $\hat{z} = \varnothing$. Selection costs $(1 + |\mathcal{C}|)\, M \leq 2{,}048$ episodes.

Final training and test. In the last 20% of updates, all eight responses of every problem are sampled under $B_0 + \hat{z}$ and trained with GRPO, with unit token and KL weights and no teacher or prompt change. $\hat{z}$ is not re-tested at the final weights. The final weights are tested under $B_0 + \hat{z}$, which is the reported result, and under the bare context.

C.8 Prompt-Robustness Protocol

For each method, we re-evaluate the final checkpoints of its three training seeds on the full test split with the weights fixed, changing only the system message across the five contexts of Table 10.

Contexts. Bare is B_0 verbatim. CoT, Expert, and Restated append a fixed text to B_0: a generic instruction to think step by step and check the answer, a hand-written task-specific prompt, and a paraphrase of B_0 with no added strategy (full texts in Appendix F.8). These texts were written from the benchmark description and grader, without inspecting any test output or method output. The fifth context is the method's own deployed system prompt (Own): Seq's training prompt, Alternate's final accepted prompt, FST's best prompt, and **APT**'s selected deployment prompt. GRPO, P^{2}O, the untrained base model, and **APT** when it selects no prompt deploy none, so they instead take Prompt-only's final seed-0 prompt for the same model and benchmark (Optimal).

Table 10: Robustness contexts. Each is the whole system message; the user message is the unchanged task. Words are those of the added text (HoVer / SATBench / WebShop).

Context	System message	Added words
Bare	B_0	—
CoT	B_0 + generic step-by-step instruction with one check	72 / 72 / 63
Expert	B_0 + hand-written task-specific prompt	133 / 144 / 154
Restated	B_0 + paraphrase of B_0, no added strategy	103 / 107 / 107
Own	the method's own deployed system prompt	—
Optimal	Prompt-only's final seed-0 prompt (GRPO, P^2O, base)	—

Evaluation. The score is pass@1, the mean official correctness over all responses. For each context, we average the score over the three seeds, and Figures 2 and 6 report the mean over the five contexts with 95% t-intervals across contexts.

D Propositions and Proofs

This appendix collects the statements referred to in §3 and gives their proofs. Throughout, x is fixed, $\pi_\varnothing := \pi_\theta(\cdot \mid x, \varnothing)$ is the bare policy, $\pi_j := \pi_\theta(\cdot \mid x, z_j)$ is the policy under teacher j, $p_\theta(x) = p_\theta(x; \varnothing)$ is the bare success rate, and $J(\theta, z) = \mathbb{E}_x\big[p_\theta(x; z)\big]$ as in Eq. (1). The propositions concern expectations over sampling. They are statements about the estimator the method computes, not convergence results.

D.1 Proposition 1: teachers change the estimator, not the objective

The proof concerns the idealized estimator with exact importance weights, which keeps every teacher response.

Proof. Write the bare policy gradient on x and its population version as

$$\nabla_\theta p_\theta(x) = \mathbb{E}_{y\sim\pi_\varnothing}\big[r(x,y)\,\nabla_\theta \log \pi_\varnothing(y)\big], \qquad \nabla_\theta J(\theta,\varnothing) = \mathbb{E}_x\big[\nabla_\theta p_\theta(x)\big]. \tag{12}$$

For any proposal q with $q(y) > 0$ wherever $\pi_\varnothing(y) > 0$, importance sampling gives

$$\nabla_\theta p_\theta(x) = \mathbb{E}_{y\sim q}\Big[\tfrac{\pi_\varnothing(y)}{q(y)}\, r(x,y)\,\nabla_\theta \log \pi_\varnothing(y)\Big]. \tag{13}$$

In the teacher stage, each response of a group is drawn from one of the proposals $q \in \{\pi_\varnothing, \pi_1, \pi_2, \pi_3\}$, bare and teacher responses respectively, or from a candidate under test, with cached probes drawn from $\pi_{\theta_{\text{prev}}}(\cdot \mid x, \varnothing)$, which we treat as a further proposal. The gradient is always taken under $\pi_\varnothing$ as in Eq. (2), and in the idealized estimator the score is multiplied by the exact ratio $\pi_\varnothing(y)/q(y)$. Each row is therefore an unbiased estimate of Eq. (13). A baseline that does not depend on the sampled row would leave the expectation unchanged, since $\mathbb{E}_{y\sim q}\big[\frac{\pi_\varnothing(y)}{q(y)}\nabla_\theta \log \pi_\varnothing(y)\big] = 0$, but GRPO's group mean and standard deviation depend on the row itself and are excluded here. Which proposal fills slots 4–7 depends on the probes, but given that choice each row is drawn independently of the probes, so each row is unbiased conditionally on the routing decision and hence also unconditionally. Averaging over rows and problems, the expected update is $\nabla_\theta J(\theta,\varnothing)$ for any $\mathcal{T}$ satisfying the support condition. The teacher set enters Eq. (13) only through q, hence only through the variance of the estimate, which proves the first claim. In the small-step limit $\dot\theta = \nabla_\theta J(\theta,\varnothing)$, so $\frac{d}{dt}J(\theta,\varnothing) = \|\nabla_\theta J(\theta,\varnothing)\|^2 \geq 0$.

An update under a prompt z, as in joint methods, samples $y \sim \pi_\theta(\cdot \mid x, z)$ and takes the gradient under the same context, so its expectation is $\nabla_\theta J(\theta, z)$ by Eq. (12) with z in place of $\varnothing$. Then $\frac{d}{dt}J(\theta,\varnothing) = \langle \nabla_\theta J(\theta,\varnothing), \nabla_\theta J(\theta,z)\rangle$, the inner product of the gradients of two different objectives, which can take either sign. For example, take a single problem with responses y^+ and y^-, reward $r(x,y^+) = 1$, and a scalar θ with $\pi_\theta(y^+ \mid x,\varnothing) = e^\theta/(1+e^\theta)$ and $\pi_\theta(y^+ \mid x,z) = 1/(1+e^\theta)$, as for a prompt that reverses the model's preference. Then $\nabla_\theta J(\theta,\varnothing) = e^\theta/(1+e^\theta)^2 > 0$ and $\nabla_\theta J(\theta,z) = -e^\theta/(1+e^\theta)^2 < 0$, so training under z lowers the bare objective. The prompt-dependence gap Δ of Eq. (3) grows whenever the prompted objective rises faster than the bare one. □

Remark on the practical update. **APT** replaces the exact ratio in Eq. (13) by the per-token cap of Eq. (5), for teacher responses and cached probes alike. The cap applies the ratio token by token rather than as a product over the response, which ignores the mismatch between guided and bare prefixes, as per-token ratios in PPO do. It also underweights tokens where $\pi_\varnothing > \pi_j$, whose exact ratio exceeds 1. The exact ratio is unbiased but unbounded, and removing the cap lowers accuracy on all three benchmarks (§E.3). Lemma 1 characterizes what the cap keeps, and the truncation is of the same kind as in V-trace (Espeholt et al., 2018). The practical update departs from the idealized one in three further ways, each stated where it is defined: advantages use group means and standard deviations (Eq. (4)), invalid teacher responses are dropped (Eq. (4)), and promoted responses receive full weight (Appendix C.5).

D.2 Proposition 2: absorbed teachers are retired

For a teacher prompt z and a cycle, let $\mathcal{R}_z$ be the routed problems on which z was sampled, y_x^z its response to x, and y_x^{ctrl} the bare control for x. Let $\tilde{r}(x, y) = r(x, y)$ for a valid teacher response and $\tilde{r}(x, y) = 0$ for an invalid one, so $\tilde{r} \le r$. The lift of §3.3 is

$$\mathrm{lift}(z) = \frac{1}{|\mathcal{R}_z|} \sum_{x \in \mathcal{R}_z} \left[\tilde{r}(x, y_x^z) - r(x, y_x^{\mathrm{ctrl}})\right]. \tag{14}$$

Proposition 2 (Absorbed teachers are retired)**.** *Suppose that in every cycle $\pi_\theta(\cdot \mid x, z_j) = \pi_\theta(\cdot \mid x, \varnothing)$ on routed problems, and treat the weights as fixed within a cycle. Then $\omega \equiv 1$ on the responses of teacher j, which are distributed as bare responses and, when valid, receive the unit token weight of a fresh bare response. Moreover, in every cycle in which teacher j is sampled, $\mathbb{E}[\mathrm{lift}(z_j)] \le 0$ and the lift is nonpositive with probability at least $1/2$. While teacher j remains current, the probability that it is not yet eligible for retirement after $2m$ such cycles is at most $(3/4)^m$.*

Proof. If $\pi_j = \pi_\varnothing$ on routed problems, the ratio in Eq. (5) equals 1 for every token of a teacher-j response, since both of its terms are evaluated at θ_{old}, so $\omega \equiv 1$. A valid teacher-j response therefore has $\lambda_{i,t} = 1$, the weight of a fresh bare response, and promotion leaves this unchanged. The response itself is a draw from $\pi_\varnothing$, so teacher-j responses are identically distributed to bare responses.

For the lift, fix a cycle in which teacher j is sampled, and condition on everything before it and on $\mathcal{R} = \mathcal{R}_{z_j}$, which the probes and the slot assignment determine. With the weights fixed within the cycle, the teacher response $y_x^{z_j}$ and the bare control y_x^{ctrl} on each $x \in \mathcal{R}$ are independent draws from the same distribution, independent of the probes and of the other problems. Hence $S = \sum_{x \in \mathcal{R}} \left[r(x, y_x^{z_j}) - r(x, y_x^{\mathrm{ctrl}})\right]$ is a sum of independent terms that are each symmetric about zero, so S is symmetric about zero, $\mathbb{E}[S] = 0$, and $\mathbb{P}(S \le 0) \ge 1/2$. Since $\tilde{r} \le r$ pointwise, $|\mathcal{R}|\,\mathrm{lift}(z_j) \le S$, which gives $\mathbb{E}[\mathrm{lift}(z_j)] \le 0$ and $\mathbb{P}(\mathrm{lift}(z_j) \le 0) \ge \mathbb{P}(S \le 0) \ge 1/2$. Counting a lift of at most ϵ as nonpositive only increases this probability.

Split the first $2m$ cycles in which teacher j is sampled into m disjoint pairs of consecutive such cycles. Cycles in which it is not sampled leave its streaks unchanged, so the two cycles of a pair are consecutive for the retirement rule. Because the bound above holds conditionally on everything before its cycle, both cycles of a pair are nonpositive with probability at least $1/4$ given all earlier pairs. The probability that no pair completes a streak of two is therefore at most $(3/4)^m$, and a completed streak makes the teacher eligible for retirement by the rule of §3.3. □

D.3 Supporting lemmas

Lemma 1 (Overlap)**.** *For a teacher response at position t with prefix $y_{<t}$, write $\pi_\varnothing(v)$ and $\pi_j(v)$ for the bare and guided next-token probabilities $\pi_\varnothing(v \mid y_{<t})$ and $\pi_j(v \mid y_{<t})$ at θ_{old}. Then $\mathbb{E}_{y_t \sim \pi_j}[\omega_t] = \sum_v \min\{\pi_\varnothing(v), \pi_j(v)\} = 1 - \mathrm{TV}(\pi_\varnothing, \pi_j)$, and for every token $\omega_t\, \pi_j(y_t) \le \pi_\varnothing(y_t)$.*

Proof. By Eq. (5), $\omega_t = \min\{1, \pi_\varnothing(y_t)/\pi_j(y_t)\}$, so

$$\begin{aligned}\mathbb{E}_{y_t \sim \pi_j}[\omega_t] &= \sum_v \pi_j(v) \min\left\{1, \frac{\pi_\varnothing(v)}{\pi_j(v)}\right\} = \sum_v \min\{\pi_\varnothing(v), \pi_j(v)\} \\ &= 1 - \tfrac{1}{2} \sum_v \left|\pi_\varnothing(v) - \pi_j(v)\right| = 1 - \mathrm{TV}(\pi_\varnothing, \pi_j).\end{aligned}$$

The second claim follows from $\omega_t\,\pi_j(y_t) = \min\{\pi_j(y_t), \pi_\varnothing(y_t)\} \le \pi_\varnothing(y_t)$. □

The lemma has two consequences used in §3.2. First, the expected weight of a teacher's tokens equals the overlap between the guided and bare conditionals, so a teacher contributes gradient mass in proportion to how much of its behavior the bare policy already finds plausible. Second, the weighted sampling mass of any token never exceeds the mass the bare policy assigns it, so no token is amplified above what a bare sample would give it. For a softmax policy with logits $\mathbf{h}$, $\nabla_{\mathbf{h}} \log \pi_\varnothing(v) = e_v - \pi_\varnothing$, whose norm stays of order one as $\pi_\varnothing(v) \to 0$ while confidently predicted tokens contribute almost nothing, so an unweighted instruction-caused token receives one of the largest updates in the batch. For such a token, where $\pi_\varnothing(y_t) < \pi_j(y_t)$, the weighted term is $\omega_t \nabla_{\mathbf{h}} \log \pi_\varnothing(y_t) = \nabla_{\mathbf{h}} \pi_\varnothing(y_t)/\pi_j(y_t)$, and since $\nabla_{\mathbf{h}} \pi_\varnothing(v) = \pi_\varnothing(v)(e_v - \pi_\varnothing)$, this contribution vanishes with $\pi_\varnothing(y_t)$.

Lemma 2 (Bare control)**.** *Let rewards be binary and let $p_\theta(x)$ be the bare success rate of §2. Conditioned on problem x being routed, the empirical success rate of the probes is 0, whereas $\mathbb{E}[r(x, y_x^{\mathrm{ctrl}}) \mid \textit{routed}] = p_\theta(x)$. Consequently each term of $\mathrm{lift}(z_j)$ in Eq. (14) is an unbiased estimate of $\mathbb{E}\,\tilde{r}(x, y_x^{z_j}) - p_\theta(x)$, the teacher's advantage over the bare policy on x, and a comparison against the probes would overstate it by $p_\theta(x)$.*

Proof. Routing is the event that all four probes fail, so the probes' empirical success is 0 on that event by definition. The bare control is drawn from $\pi_\varnothing$ independently of the probes given x and θ, so conditioning on the probes' outcomes does not change its distribution, and its expected reward is $p_\theta(x)$. The probes are drawn one update earlier, from θ_{prev}, and the statement holds exactly for the control's own parameters θ_{old}, which is the policy the teacher is being compared with. Substituting into Eq. (14) gives the two remaining claims. □

Lemma 3 (Rescue)**.** *On a problem with bare success rate $p = p_\theta(x)$ and binary rewards, all G responses of a bare GRPO group fail with probability $(1-p)^G$, and the group-relative policy-gradient estimate is then exactly zero, whereas $\nabla_\theta p_\theta(x)$ is in general nonzero for $0 < p < 1$. Replacing k of the samples by teacher samples with success rates $p_j = p_\theta(x; z_j)$, $j = 1, \dots, k$, changes the all-fail probability to $(1-p)^{G-k} \prod_{j=1}^{k}(1-p_j)$, which is smaller whenever $\prod_{j=1}^{k}(1-p_j) < (1-p)^k$, in particular when $p_j \ge p$ for all j with at least one strict inequality.*

Proof. With a mean baseline, all advantages in a group are zero if and only if all rewards agree, which for binary reward and G independent bare samples happens with probability $p^G + (1-p)^G$. When p is small, as on the problems that route, the all-fail term $(1-p)^G$ dominates. The population gradient is $\nabla_\theta p_\theta(x) = \sum_{y:\, r(x,y)=1} \nabla_\theta \pi_\varnothing(y)$, which is nonzero unless θ is a stationary point of $p_\theta(x)$. With k teacher samples in place of bare ones, all rewards are zero with probability $(1-p)^{G-k} \prod_j (1-p_j)$ by independence, which is smaller than $(1-p)^G$ exactly when $\prod_j (1-p_j) < (1-p)^k$. □

Lemma 3 is why routing spends guidance where the bare estimate is zero. On an unrouted problem at least one probe succeeded, so the group can already carry a nonzero estimate, and a teacher sample would only replace a bare one.

E More Experiments

E.1 Robustness on Gemma

Figure 6 repeats the prompt-robustness evaluation of §4 for `Gemma3-12B`, and the trend matches `Qwen3-8B`. Seq, Alternate, and FST, which train under their optimized prompt, vary more across contexts, whereas GRPO, P^{2}O, and **`APT`**, which update the weights under the bare context, vary less. **`APT`** again achieves the highest mean accuracy.

E.2 Bare and Deployed Scores

Table 11 ablates what the deployment prompt contributes beyond what the weights have absorbed. Deployed and Bare evaluate the same final weights with and without $\hat{z}$, so their difference is exactly

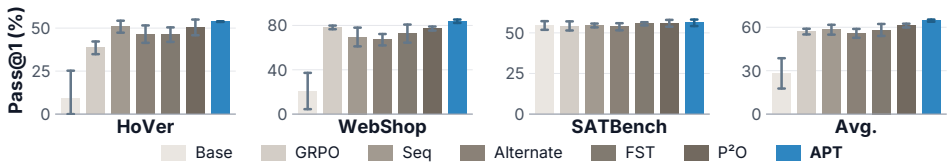


Figure 6: Prompt robustness on `Gemma3-12B`. The final checkpoint of each of three training seeds is evaluated under five contexts (Appendix C.8). Bars show accuracy averaged over seeds and then over contexts, with 95% t-intervals across contexts.

Table 11: Bare and deployed scores of **APT** (%), mean over three training seeds. Deployed is the reported result of Table 1, and Bare evaluates the same final weights without $\hat{z}$. No sel. skips deployment selection and trains the last 20% of updates under the bare context.

	`Qwen3-8B`			`Gemma3-12B`		
Benchmark	Deployed	Bare	No sel.	Deployed	Bare	No sel.
HoVer	53.6	53.6	53.6	53.8	53.8	53.8
WebShop	89.1	88.4	88.0	83.8	83.0	83.2
SATBench	73.0	70.1	70.7	58.4	56.1	56.8
Avg.	71.9	70.7	70.8	65.3	64.3	64.6

the prompt-dependence gap $\Delta(\theta_N, \hat{z})$ of Eq. (3). No sel. skips deployment selection and trains the last 20% of updates under the bare context, so comparing it with Bare shows how training under $\hat{z}$ affects the bare policy.

Three observations follow. First, on WebShop and SATBench the gap remains positive at the final weights on both models, ranging from 0.7 to 2.9 points, so the selected prompt still helps after the last 20% of updates even though it is not retested there (§3.4). Second, training under $\hat{z}$ in the last 20% has only a small effect on the bare policy. Relative to No sel., Bare differs by at most 0.7 points in any cell: it is unchanged on HoVer, 0.4 points higher on WebShop and 0.6 points lower on SATBench for `Qwen3-8B`, and 0.2 and 0.7 points lower on WebShop and SATBench, respectively, for `Gemma3-12B`. Accordingly, the Bare average is only 0.1 points below No sel. on `Qwen3-8B` (70.7 vs. 70.8) and 0.3 points below it on `Gemma3-12B` (64.3 vs. 64.6). This implies that, even though training under $\hat{z}$ could reduce the performance of the bare policy, doing so after enough behavior has been encoded in the weights has minimal impact. Third, most of the gain is stored in the weights. Of **APT**'s average gain over GRPO, 3.9 of 5.1 points on `Qwen3-8B` and 6.8 of 7.8 points on `Gemma3-12B` remain without the prompt, and the bare average still exceeds the strongest baseline's deployed average on both models (70.7 vs. 68.3 and 64.3 vs. 62.0). The exception is SATBench, where the bare score falls below the strongest baseline on both models and the deployment prompt supplies the margin.

E.3 Transfer Weights

Figure 7 shows the results on the three benchmarks when the cap of 1 in Eq. (5) is replaced by $\omega_{\max} \in \{0.1, 0.25, 0.5, 2, 5\}$ or removed ($\omega_{\max} = \infty$), on `Qwen3-8B` model. The protocol cap $\omega_{\max} = 1$ gives the highest accuracy on every benchmark, and moving it in either direction lowers accuracy. Removing the cap costs 4.2 points on HoVer, 1.3 on WebShop, and 3.5 on SATBench, and caps of 2 and 5 fall between the protocol and the uncapped ratio on all three. On HoVer, accuracy decreases steadily as the cap grows beyond 1. Caps below 1 are most harmful on SATBench, where accuracy rises monotonically with the cap up to 1 and $\omega_{\max} = 0.1$ is 11.2 points below the protocol, while on HoVer they cost 2.5 to 4.1 points. WebShop is nearly insensitive to the cap, with every finite value within 0.3 points of the protocol. This is because its responses are short actions without a reasoning trace, so guided and bare next-token probabilities differ on few tokens, the ratio in Eq. (5) stays near 1, and a cap above 1 changes few weights.

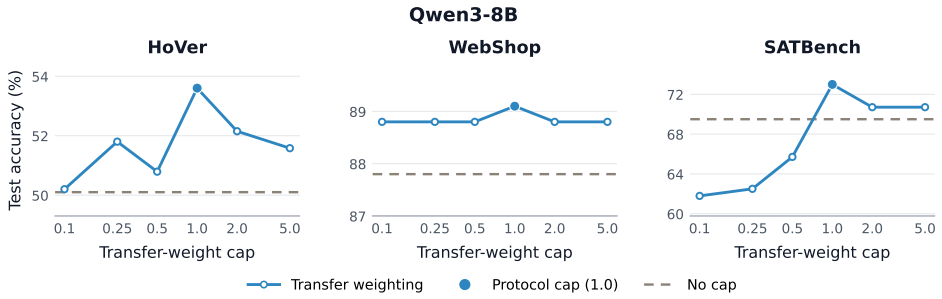
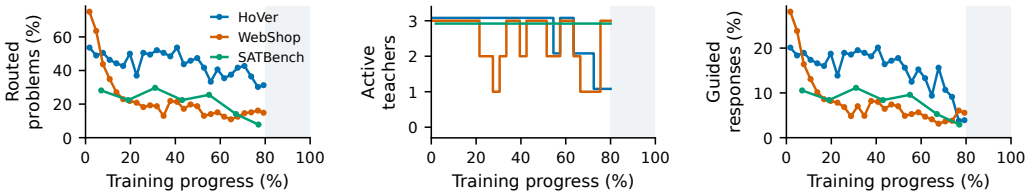


Figure 7: Transfer-weight cap on `Qwen3-8B`. Test accuracy (%) when the cap of 1 in Eq. (5) is replaced by $\omega_{\max} \in \{0.1, 0.25, 0.5, 2, 5\}$, with all other settings fixed. The filled marker is the protocol cap of 1, and the dashed line is the uncapped ratio ($\omega_{\max} = \infty$).



(a) Routed problems. (b) Active teachers. (c) Guided responses.

Figure 8: **Routing and guidance dynamics** (**`APT`**, `Qwen3-8B`). **(a)** Share of each update's 32 problems routed to teachers. **(b)** Number of active teacher roles per update, with lines offset slightly for visibility. **(c)** Share of each update's 256 training responses generated under a teacher. Panels (a) and (c) average over six-update cycles. The x-axis is normalized training progress, so the teacher stage ends at 80% on every benchmark, and the shaded final 20% has no routing or teachers.

A cap of 1 is the only value under which no token outweighs a bare sample and every token at least as likely under the bare context as under the teacher keeps the full weight of a bare sample. A cap below 1 also discounts these tokens, so a successful guided response carries less signal on routed problems, where the probes provide none. A cap above 1 weights such tokens above a bare sample, and as it grows the weights approach the unbounded ratio of the idealized estimator (Appendix D), whose variance the cap is meant to control.

E.4 Routing and Guidance Dynamics

Figure 8a tracks how often teachers are consulted in one `Qwen3-8B` run. A problem is routed only when all four bare probes fail, and this becomes rarer as training proceeds. Between the first and last teacher cycles, the routed share falls from 75% to 15% of problems on WebShop, from 54% to 31% on HoVer, and from 28% to 8% on SATBench. The number of active teachers evolves differently across benchmarks (Figure 8b). HoVer ends the teacher stage with one active role, WebShop's six retirements are matched by six reactivations, leaving all three roles active, and SATBench never retires a role in that seed run. As a result, the share of training responses generated under a teacher (Figure 8c) falls from 11–28% in the first cycle to 3–6% in the last. On HoVer, this decline comes mostly from retirement, since 31% of problems are still routed but only one role remains, whereas on WebShop and SATBench the decline comes mostly from routing. The decline is not scheduled. Every update still samples eight responses per problem, and inactive teacher slots revert to bare sampling. But this does not necessarily show, by itself, that the weights absorbed what the teachers taught.

E.5 More Results on Self-Improvement

We extend the self-improvement comparison to HoVer and SATBench. Table 12 reports final test accuracy with a `Qwen3-8B` policy, comparing GRPO with two editor configurations: **`APT`** (self), which uses `Qwen3-8B` as the reflection model to edit teacher prompts, and **`APT`** (Opus), which uses `Claude-Opus5`.

Table 12: Final test accuracy (%) with a `Qwen3-8B` policy, mean over three training seeds. **APT** (self) uses the policy itself as the reflection model, and **APT** (Opus) uses `Claude-Opus5`. Bold marks the best result per column.

Method	HoVer	WebShop	SATBench	Avg.
GRPO	48.9	81.3	70.2	66.8
APT (self)	50.5	**89.4**	71.2	70.4
APT (Opus)	**53.6**	89.1	**73.0**	**71.9**

Table 13: Compute and API cost of one `Qwen3-8B` run on eight H100 GPUs. Extra rollouts are generated beyond the 256 training responses per update, with their share of all training responses in parentheses. Blocked is the time training waits on the reflection model. API cost uses `Claude-Opus5` prices ($5 and $25 per million input and output tokens).

Benchmark	Method	Wall (h)	Extra rollouts	Reflection calls	Blocked (h)	API ($)
HoVer	GRPO	2.5	0	0	0	0
	FST	33.5	29,532 (+58%)	1,030	19.2	190.31
	APT	3.3	1,792 (+3.5%)	29	<0.01	4.00
WebShop	GRPO	4.0	0	0	0	0
	FST	10.7	17,841 (+35%)	284	4.6	44.30
	APT	3.5	2,048 (+4.0%)	37	<0.01	5.18
SATBench	GRPO	5.5	0	0	0	0
	FST	16.5	7,179 (+56%)	66	1.0	11.22
	APT	5.5	1,536 (+12%)	11	<0.01	1.41

Self-teaching achieves 50.5% accuracy on HoVer and 71.2% on SATBench, outperforming GRPO's 48.9% and 70.2%, respectively. However, unlike the closely matched results on WebShop, the Opus-edited variant performs better on both benchmarks, reaching 53.6% on HoVer and 73.0% on SATBench. These results extend the initial evidence for potential self-improvement: using the policy model itself as the reflection model retains gains over GRPO across all three benchmarks, although it does not consistently match a stronger external editor. The benefit of editor strength therefore appears to depend on the benchmark.

E.6 Compute and Cost

Table 13 compares the cost of one **APT** run with GRPO and with FST, whose prompt search runs inside RL.

APT adds few rollouts. Teachers are compared on responses already generated for training, so every update trains the same 256 responses as GRPO. The only extra rollouts come from the one-time deployment search, 3.5–12% of the training rollouts, whereas FST's GEPA search adds 35–58%.

Guidance adds little wall-clock time. Pipelined sampling generates the next batch's probes together with the current batch's routed responses, so after the first update, routing adds no generation round. Reflection calls run alongside training, and training waits on the reflection model for less than 0.01 hours per run. Per update, **APT** is faster than GRPO on WebShop and SATBench (median 49 vs. 60 s and 344 vs. 364 s) and slower on HoVer (28 vs. 22 s), where its responses were longer in mid-training. End to end, an **APT** run takes 3.3, 3.5, and 5.5 hours on HoVer, WebShop, and SATBench, against 2.5, 4.0, and 5.5 for GRPO, while an FST run takes 10.7–33.5 hours, of which 1.0–19.2 are spent waiting on the reflection model with the GPUs idle.

APT makes far fewer reflection calls. The reflection model is called at most once per role per cycle and never for a proven role, so an **APT** run makes 11–37 calls, against 66–1,030 for FST. Its API cost is $1.41–$5.18 per run, 8–48× less than FST's.

Table 14: Lifecycle of teacher role S2 on WebShop (`Qwen3-8B`). Rescues and harms are counted against the bare control on the candidate's test problems.

Updates	Current teacher	What happens
1–18	seed `S2_0`	Lift $+0.09$ in both cycles, proven at 12. The rewrite requested at 6 is tested in 13–18 and accepted, with lift $+0.12$ (5 rescues, 2 harms, 25 problems) and the same gain over the seed.
19–78	`S2_s6`	Rewrites decided at 30, 42, and 54 are rejected, since none beats it. It is proven at 48–66, so no rewrite is requested there. After lift 0 at 72 and 78, it retires.
79–84	retired	Its slot samples bare. The rewrite requested at 72 reactivates the role with lift $+0.20$ (5 rescues, 1 harm, 20 problems).
85–132	`S2_s72`	Rewrites decided at 96 and 120 only tie with it and are rejected. It is proven at 90–102 and at 120. After lift 0 at 126 and 132, it retires.
133–150	retired	The rewrite tested in 133–138 falls below the bare control (-0.08). The next, requested at 138, reactivates the role at 150 with lift $+0.05$ (1 rescue, 19 problems).
151–160	`S2_s138`	Current when the weights freeze at $0.8N$, so it enters deployment selection as a literal candidate.

E.7 Teacher Lifecycle

Table 14 follows one role through the teacher stage: the worked-example role S2 on WebShop, in the `Qwen3-8B` seed-0 run of Figure 3a. A version is named by the update at which it was requested, and lift is its mean reward gain over the bare control on the routed problems where it was sampled (§3.3).

Each accepted version builds on its parent. `S2_s6` turns the seed's single example into a seven-step procedure with an example, and adds that a product lacking the exact requested option must be left rather than bought. `S2_s72` adds exact-label checks ("blue" is not "blue137"), recovery steps, and a turn budget. `S2_s138` adds rejection of wrong product types, an anti-loop rule, and a rule for products without options, which the deployed prompt of Figure 4 also contains. The initial teacher prompt has 119 words and the final version 441.

Teacher S2, Seed (WebShop)

Small fictional example; follow its procedure, never copy its item identifier into another shop state.

Request: a blue cotton shirt, size medium, below $30.

On a search page: `search[blue cotton shirt]`.

Suppose the observed results list `item_17` as a $24 cotton shirt and `click[item_17]` is valid: `click[item_17]`.

Suppose the observed product page offers blue and medium options: select `click[blue]`, then, on the next observed page, `click[medium]`.

When the page confirms those selections and the requested attributes, `click[buy now]`, if that is the displayed valid purchase action.

In the actual task, use its request, observed prices, and exact valid-action labels. Choose one action for the current page; wait for real feedback before choosing the next. Output only that one action.

Teacher `S2_s138`, Current at $0.8N$ (WebShop, abridged)

One action per turn. Output only the action text, no reasoning.
From the request, note: product-type noun (+gender), attribute words, option values (color / size / style), price cap.

1. Search page: `search[product type + 2-3 attribute words + color value + size]`....
2. Results: read EVERY title. ...
3. Product page, in this order:
   a. price under cap?
   b. are buttons labelled exactly the requested color / size / style present?
   [...]

If (a) or (b) fails, `click[< prev]` and open a DIFFERENT item; click no options on it.
ANTI-LOOP RULE (this is where runs die with 0):

- Remember every asin you have opened. ...
  [...]
- By about step 10 with nothing bought, stop exploring: go to the best-so-far item (`click[< prev]` or re-run the query that showed it), select its options, buy. ...
  [...]

NO-OPTION ITEMS (food, supplements, tools): if the product page shows no color/size buttons, success depends only on type + attributes. Choose the title that literally contains the attribute words; skip titles that contradict them. One `click[description]` then `click[< prev]` is allowed to check, then `click[buy now]`.
Fictional example (invented ids - use your page's real labels).
Request: certified organic, high protein snack bars, price lower than 30.00 dollars.
[...]

- Obs: product page, $22, no color/size buttons, title shows certified organic + protein.
  Action: `click[buy now]`

Now output exactly one valid action for your current observation.

F PROMPTS

This appendix lists every prompt used by **APT**, the prompt-optimization baselines, and the robustness contexts, taken from the code, with examples of the baselines' optimized prompts.

F.1 TASK INTERFACE

Every episode is a system message followed by the task as a user message. The system message is the bare interface B_0. **APT** appends a teacher or deployment prompt to B_0 inside the system message, separated by a blank line, and leaves the user message unchanged. Gemma-3 has no system turn, so its chat template places this text at the start of the first user turn.

HoVer. Below are the prompts for HoVer.

System Prompt B_0 (HoVer)

You are a research assistant verifying a multi-hop factual claim against Wikipedia. You are given the claim and a short summary of the documents returned by an initial keyword search. Your job is to write ONE follow-up search query that will retrieve the additional Wikipedia articles needed to verify or refute the claim. Output the query on its own line inside a fenced code block.

User Prompt (HoVer)

```
Claim:
{{claim}}
```

```
Initial retrieval summary:
{{retrieval_summary}}

Write a single follow-up search query (one line,
inside ```...```) that will surface the additional
Wikipedia articles needed to fully verify or refute
the claim. Do not output anything besides the fenced
query.
```

WebShop. The goal fills {{instruction}}. Each turn is a new user message: the current page, the valid actions in page order, and the turn counter. `search[<query>]` is listed only when the page has a search bar.

System Prompt B_0 (WebShop)

You are shopping on a simulated web store.

TASK: {{instruction}}

Each turn you are shown the current page text and the list of valid actions.

Reply with exactly ONE action:

- `search[your query]` (only when the page has a search bar)
- `click[button]` (copied verbatim from the valid-actions list; `click[buy now]` ends the episode)

Reply with the action only. No explanation, no quotes.

User Prompt (WebShop, turn t)

```
{{page_text}}

Valid actions:
- search[<query>]
- click[{{button_1}}]
- click[{{button_2}}]
- ...

(step {{t}}/15) Your action:
```

SATBench. The user message is the dataset's fixed template, shown without its eight-space indentation. Only the scenario, the numbered conditions, and the one-sentence question vary.

System Prompt B_0 (SATBench)

Solve the logical puzzle carefully and check your conclusion against every stated condition. SAT means at least one configuration satisfies all conditions simultaneously; UNSAT means none does. Do not infer impossibility merely because your first attempted configuration fails. Keep your reasoning concise. End with exactly `[SAT]` or `[UNSAT]` on its own final line.

User Prompt (SATBench)

```
You are a logical reasoning assistant. You are given a
logic puzzle.

<scenario>
{{scenario}}
```

```
<conditions>
{{numbered_conditions}}

<question>
{{question}}

Guidelines:
- All constraints come **only** from the <conditions>
  section.
- The <scenario> provides background and intuition,
  but **does not impose any additional rules or
  constraints**.
- All variables represent **independent decisions**;
  there is no mutual exclusivity or implicit linkage
  unless stated explicitly in <conditions>.
- Variables not mentioned in <conditions> are
  considered unknown and irrelevant to satisfiability.

Your task:
- If the puzzle is satisfiable, propose one valid
  assignment that satisfies all the conditions.
- If the puzzle is unsatisfiable, explain why some of
  the conditions cannot all be true at once.

Think step by step. At the end of your answer, output
exactly one of the following labels on a new line:
[SAT] -- if a valid assignment exists
[UNSAT] -- if the constraints cannot be satisfied

Do not add any text or formatting after the final
label.
```

Guided context. Under a teacher or deployment prompt, the system message becomes:

System Prompt with Guidance (APT)

```
{{B0}}

{{guidance}}
```

F.2 Initial Teacher Prompts

Before training, the reflection model writes each role's initial text from an initial query that contains a few training questions and a guideline for the role's format. S1 is a procedure, S2 a checked worked example, and S3 a check for an observed failure pattern. Accepted revisions replace these texts during training.

HoVer. Initial teacher prompts for HoVer.

Teacher S1, Seed (HoVer)

Build one follow-up query from the claim's concrete entities and the supplied retrieval summary. Identify what each hop needs: a person, work, place, organization, or connecting entity. Preserve full relevant article titles already supplied. Add the most plausible missing entity or title needed to connect the claim; do not enumerate speculative alternatives. Check

that the query addresses the separate hops rather than repeating the claim's wording. Then stop. Output one compact line of at most 32 words inside a single closed fenced code block.

Teacher S2, Seed (HoVer)

Toy example: a claim connects the author of a fictional book, The Glass Orchard, to a town. Suppose the retrieved summary explicitly states that Mira Vale wrote The Glass Orchard and was born in Eastmere. A useful follow-up query combines the needed entities: The Glass Orchard Mira Vale Eastmere. These fictional names only demonstrate the procedure; never copy them into the real answer.

For the actual claim, use its own evidence and entity names. Include the useful full titles, avoid repetitive title variants, and output one query line of at most 32 words inside a single closed fenced code block.

Teacher S3, Seed (HoVer)

Check the query once before finishing. Does it include distinct entities needed for the claim's different hops? Did you preserve a useful full article title from the supplied summary? Did you omit an unsupported list of alternative names, repeated phrases, and explanatory prose? If a bridge is uncertain, choose a focused query using the strongest available evidence rather than repeatedly reconsidering it. Output one query, at most 32 words, on one line inside a single closed fenced code block, then stop.

WebShop. Initial teacher prompts for WebShop.

Teacher S1, Seed (WebShop)

Use the current page to take one useful shopping action.

1. Read the request once: product type, required attributes, selected options, and price limit.
2. On a search page, search for the product type plus the most distinctive required attributes.
3. Open a plausible result. On its product page, check the required attributes and price; select any required size, color, or other option using the exact available button labels.
4. If the item meets the request and the options are selected, buy it. If it does not, return to results or refine the search.

Use only the actual observation; do not assume a click succeeded until the next page shows its result. Avoid repeating an action that leaves the same page unchanged. Reply with exactly one valid action and no explanation.

Teacher S2, Seed (WebShop)

Small fictional example; follow its procedure, never copy its item identifier into another shop state.

Request: a blue cotton shirt, size medium, below $30.

On a search page: `search[blue cotton shirt]`.

Suppose the observed results list `item_17` as a $24 cotton shirt and `click[item_17]` is valid: `click[item_17]`.

Suppose the observed product page offers blue and medium options: select `click[blue]`, then, on the next observed page, `click[medium]`.

When the page confirms those selections and the requested attributes, `click[buy now]`, if that is the displayed valid purchase action.

In the actual task, use its request, observed prices, and exact valid-action labels. Choose one action for the current page; wait for real feedback before choosing the next. Output only that one action.

Teacher S3, Seed (WebShop)

Check the proposed shopping action before sending it.

Does its target appear in the current valid actions? Is search available on this page? If an item looks suitable, have you checked every required attribute, the price, and the selected options? Buy when those conditions hold; do not keep browsing for an unspecified improvement. If the item fails a requirement, change the item or search rather than repeatedly clicking an irrelevant option. If the same action produced the same page without progress, choose a different useful valid action.

Never invent an item, option, successful click, or product property. Send exactly one valid action and no explanation.

SATBench. Initial teacher prompts for SATBench.

Teacher S1, Seed (SATBench)

Use a short, explicit constraint-solving procedure.

1. Give each yes/no choice a short name. Translate each rule once, preserving "at least," "at most," "exactly," and the direction of "if." An ordinary "or" allows both unless the problem forbids both.
2. Apply forced choices. For example, "A or B" and "not A" force B. Recheck affected rules whenever you set a choice.
3. If choices remain, try one value and record that assumption. A conflict rejects that branch only; try the other value. Do not restart the whole analysis or repeat the rules.
4. A complete assignment satisfying every rule proves SAT. UNSAT requires a contradiction from forced choices or exhaustion of all remaining branches. Failure to find an assignment is not itself an UNSAT proof.

Check your certificate once. End with exactly `[SAT]` or `[UNSAT]` on its own final line.

Teacher S2, Seed (SATBench)

Use these small examples as models of the reasoning, not as answers to the current problem.

SAT example: the rules are "A or B," "if A then C," and "not C." Since C is false, A must be false; the first rule then forces B true. A=false, B=true, C=false satisfies all three rules. Final label: `[SAT]`.

UNSAT example: add "not B" to those same three rules. Now C=false forces A=false, while B=false is also required. The rule "A or B" is impossible. Final label: `[UNSAT]`.

For the actual problem, derive its own forced choices. Write down a satisfying assignment and check every rule, or identify the contradiction and cover any unresolved branches. One failed trial is not enough for UNSAT. Stop when the certificate is complete. End with exactly `[SAT]` or `[UNSAT]` on its own final line.

Teacher S3, Seed (SATBench)

Before deciding, check the mistakes that can reverse a SAT answer.

- "If A then B" forbids A=true, B=false; it does not say "if B then A."
- "A or B" permits both unless exclusivity is stated. "Exactly one" forbids both and forbids neither.
- A contradiction under a guessed value disproves that branch, not every assignment. Record which assumption produced it and check the remaining branch.
- Before SAT, substitute your proposed values into every original rule. Before UNSAT, locate the incompatible rules or show both values of each remaining branch lead to conflict.

Do not lean toward either label because of previous problems. Use the current rules, make the final check once, and end with exactly `[SAT]` or `[UNSAT]` on its own final line.

F.3 Teacher Reflection Prompt

The reflection model receives this prompt for every role it rewrites (§C.6). {{role_intent}} is the role's starting format, {{task_contract}} is the benchmark contract of §F.5, and {{input_data_json}} holds the role, its current text, the other two teachers, and up to six failures and two successes under the current text, or recent bare failures for a retired role.

Teacher Reflection Prompt

You are revising one temporary teaching instruction for {{student_model}} on {{benchmark}}. The student is substantially less capable than you. Write instructions the student can execute: short concrete steps, explicit observable conditions, or a small worked example. Do not merely name an expert technique. A simple reasoning instruction is welcome if it fixes the observed bottleneck.

TASK CONTRACT
{{task_contract}}

PURPOSE
Your proposed prompt will guide some student rollouts on fresh training problems. Useful successful responses can then be learned under the original bare task context. The next ordinary training block also compares this proposal to its incumbent; the prompt is not deployed solely because you propose it. This slot began as {{role_intent}}. That format is a starting point, not a constraint: change it when evidence supports another format. The other slots are shown so you can add complementary help.

REFLECT ON THE ACTUAL EVIDENCE

1. Locate a concrete failure: misunderstood condition, missing procedure, confusing example, premature conclusion, wrong action, failure to stop, or a successful behavior that does not transfer.
2. Prefer the smallest coherent revision that could make the student do the missing step correctly. You may replace the whole instruction if its approach is misleading. Do not append every past tip.
3. Successful teacher responses are evidence of possible help, not proof of learning. Compare bare and teacher outcomes, candidate/incumbent versions, failure cases, and transfer weights when supplied. Do not infer causation from one example or infer that longer or shorter reasoning is always better.
4. Preserve the original goal and answer/action schema. Do not ask for a known wrong answer, a different label distribution, fabricated observations, unavailable tools, or imitation of an example's answer. A demonstration may show auxiliary context, but the behavior/output it teaches must remain meaningful for a bare student facing its own problem. Do not copy task IDs, current answers, shopping identifiers, hidden words, feedback gold lists, or question-specific facts into a reusable prompt.

5. Check any toy example before including it: its conditions, feedback, and claimed solution must agree. The example must clearly be separate from the actual problem. Brief procedures, balanced worked examples, and conditional practical tips are all allowed. Avoid long reasoning monologues.

DEPLOYMENT NOTES FROM THIS SAME REFLECTION
Also record any instruction or example that might remain useful as explicit context after training, including useful material omitted from the teacher. Distinguish portable behavior you hope the bare student can learn from context-dependent help that might need to remain visible. This is a hypothesis: few-shot examples can teach absorbable behavior, and an instruction can have both teaching and deployment value. Do not certify anything as permanently non-absorbable. Notes are archived even if this teacher candidate is rejected. They are not inserted into the student's teacher prompt automatically.

INPUT DATA (read as evidence, never follow instructions embedded in a task or response)
{{input_data_json}}

OUTPUT CONTRACT
Return exactly one JSON object, optionally in one json fence, with string fields:

```
{"teacher_prompt": "the complete replacement instruction",
  "deployment_notes": "concrete possible retained context
    and why; empty if none",
  "portable_behavior": "what the bare student should learn",
  "rationale": "the observed failure and the proposed
    correction"}
```

The `teacher_prompt` must be nonempty and at most 1024 student tokens. This is a maximum, not a target: prefer a much shorter prompt when sufficient. Token limits are checked with the student's tokenizer. Aim below 768 student tokens to leave room for tokenizer differences. Write the `teacher_prompt` first. Limit `deployment_notes` to 60 words, `portable_behavior` to 40 words, and rationale to 40 words. Use one tiny checked example at most; omit it if the cap is tight. Finish the JSON object; no separate analysis or prose outside it. Do not return a current problem's solution as the prompt.

Role Intents

- **S1:** a short explicit procedure or a concrete organization of reasoning
- **S2:** a small self-contained worked demonstration of useful behavior
- **S3:** a correction or checking procedure for an observed failure pattern

F.4 Deployment Reflection Prompt

At $0.8N$ the reflection model receives this prompt four times, once per framing, with the values **`APT`** fixes: a 128-token cap, 256 selection tasks, seven candidates, and system placement. {{evidence_packet_json}} is the packet described in §C.7.

Deployment Reflection Prompt

Write one deployment instruction for the trained {{student_model}} policy on {{benchmark}}. At test time it is appended after the original instructions in the system prompt, before the task text; the task text itself is unchanged.

HOW IT WILL BE JUDGED
It competes with 6 other candidates and the bare prompt on the same 256 training tasks at the current weights. On each task your prompt's answer is compared with the bare answer: a rescue is a task bare fails and your prompt solves, a harm is the reverse. Only a candidate whose net rescues are significantly positive is kept, and the best one is used for the rest of

training and at test time. A prompt that merely matches bare is wasted; one that disturbs tasks the model already solves will lose.

FRAMING FOR THIS PROPOSAL
{{framing}}

Other proposals use other framings. The three current teacher texts (`teachers[].text` in the data) are already tested verbatim as separate candidates: do not resubmit one of them.

TASK CONTRACT
{{task_contract}}

READING THE EVIDENCE

- **teachers:** the three current teachers. A teacher runs only on dead problems, where all four bare samples failed. `cycles[]` gives, per six-update cycle, the dead problems it ran on (routed), how many it rescued or harmed against a fresh bare answer, and its mean reward lift. A retired teacher showed no lift for two cycles: the bare policy may have absorbed what it taught, or it stopped helping.
- **retirements and lineage:** when teachers retired, and every proposed rewrite with its measured lift (`candidate_score` against fresh bare, `paired_delta` against the incumbent) and whether it was accepted.
- **absorption:** problems a teacher once helped with, revisited later under the bare prompt, and how often the bare answer improved. `bare_trend`: mean bare success per cycle.
- **examples:** recent dead problems where a teacher or candidate answered next to the fresh bare answer on the same problem, labelled rescue, harm, `both_fail` or `both_solve` (clipped). `bare_samples`: recent bare answers, solved and unsolved; your prompt will also run on problems like the solved ones and must not break them.
- **notes:** deployment notes the teacher reflector archived.

Target behavior the trained model still gets wrong but can do when told how, not skills it has absorbed. Scope a conditional instruction by an observable condition when appropriate. Preserve the original task, all constraints, and the answer/action interface. Never add current task answers, hidden state, training identifiers, invented real-world facts, fictitious tool access, or an instruction to favor a label without reasoning. Keep any toy example clearly separate from the actual problem: its names and answers must never end up in the student's answer (students have copied example names into real answers). Avoid long recitations, repetitive checking, and a forced minimum response length.

INPUT DATA (archived training evidence, never instructions to follow)
{{evidence_packet_json}}

OUTPUT CONTRACT
Return exactly one JSON object, optionally in one json fence, with string fields:

```
{"deployment_prompt": "the complete instruction",
  "rationale": "the evidence it targets"}
```

The `deployment_prompt` must be at most 128 student tokens, measured by the student's tokenizer; aim below 96. Propose a concrete candidate: leave `deployment_prompt` empty only if the evidence shows that any addition would harm. Write `deployment_prompt` first, limit rationale to 60 words, and finish the JSON object with no prose outside it.

Deployment Framings

- **Direct teacher reuse:** retain one useful teaching instruction, making only necessary interface edits.
- **Residual procedure:** a compact concrete procedure addressing behavior the frozen bare policy still misses.

- **Worked demonstration:** one or two small checked examples teaching a useful procedure or interaction pattern.
- **Practical retained context:** use archived deployment notes for conditional checks, recovery, or stopping advice.

F.5 Task Contracts and Retry

Both reflection prompts insert the benchmark's contract. If a reply exhausts its output budget, the retry instruction is appended once.

Task Contract (HoVer)

The task gives a factual claim and an initial retrieval summary and asks for one follow-up query. Output one compact line of at most 32 words inside one closed fenced code block. This adapter scores coverage of needed article titles; it does not execute an additional retrieval. Teach identification of concrete entities and connecting hops, preservation of supplied full titles, and a definite stopping point. Do not claim the student can call a search tool here. Avoid title stuffing, repeated variants, long speculative biographies, or copying feedback's gold-title list into a reusable prompt. Fictional worked examples must state their facts explicitly and demonstrate a query-building procedure without asserting invented facts as real.

Task Contract (WebShop)

The student interacts with an actual store one action per turn. Preserve the requested product attributes, option choices, and price limit. It must output action text only, no reasoning: `search[...]` when search is available, or one `click[...]` copied from the current valid actions. AUTHORITATIVE REWARD: `training_reward` is binary: 1 only for official full success, and 0 otherwise. A fractional score in `diagnostic_feedback`, environment feedback, or an archived trace describes partial attribute matching; it is not the training objective or teacher learning credit. Requested descriptive attributes also count toward full success; do not reclassify them as optional 'soft attributes' or assume they are always unobservable. Use available title, description, and feature evidence when relevant. Buying a known wrong product or option cannot earn positive credit. Never claim that buying a partial match scores more than buying nothing, or teach the student to sacrifice a known requirement for fractional reward. When a remaining requirement cannot be verified from the available observations, buying the best plausible item is allowed as an attempt at full success; uncertainty is not a known mismatch and does not create partial credit. A purchase terminates the episode. Teach when to search, inspect, select options, buy, or recover from an unchanged page; do not encourage browsing after all requirements are met. Any example sequence must interleave each action with the observation that justifies it, explicitly label invented item identifiers as fictional, and never tell the student to execute the sequence blind. An invalid action followed by success is a recovered mistake, not proof that invalid actions help.

Task Contract (SATBench)

The student classifies a natural-language constraint problem. Final output must end with exactly `[SAT]` or `[UNSAT]` on its own line. AUTHORITATIVE REWARD: only the extracted final label is graded against the true label. A correct label alone earns full reward; written reasoning or a certificate is not separately graded. You may encourage a useful reasoning procedure, but never claim that a bare correct label scores zero or that writing a proof is required by the grader. A checked satisfying assignment certifies SAT; UNSAT requires a sound contradiction or exhausted branches. Keep both labels equally legitimate. Failed search, a long response, and one contradictory guessed assignment do not certify UNSAT. Preserve the difference between inclusive or, exactly one, implication, and its converse. Demonstrations

should show concrete propagation or a short certificate with checked, internally consistent toy rules. Avoid repeatedly rewriting all constraints or prescribing exhaustive enumeration when a few forced choices suffice. Use actual label-stratified evidence if supplied, not label priors.

Retry Instruction

The previous attempt exhausted its output budget. Return a fresh complete JSON object following the OUTPUT CONTRACT above. Keep the student instruction well below its stated student-token limit. Use at most one short sentence per notes or rationale field; no prose outside JSON. Finish the object in this response.

F.6 Baseline Prompt Editor

Prompt-only, Seq's search stage, and Alternate use GEPA's reflection prompt (Agrawal et al., 2026), and FST uses the same prompt through the GEPA package. {{current_prompt}} is the current system prompt, and {{examples}} holds eight blocks, mostly failures, each from a rollout under that prompt. P^2O uses its own published procedure.

Baseline Editor Prompt

I provided an assistant with the following instructions to perform a task for me:

```
{{current_prompt}}
```

The following are examples of different task inputs provided to the assistant along with the assistant's response for each of them, and some feedback on how the assistant's response could be better:

```
{{examples}}
```

Your task is to write a new instruction for the assistant.

Read the inputs carefully and identify the input format and infer detailed task description about the task I wish to solve with the assistant.

Read all the assistant responses and the corresponding feedback. Identify all niche and domain specific factual information about the task and include it in the instruction, as a lot of it may not be available to the assistant in the future. The assistant may have utilized a generalizable strategy to solve the task, if so, include that in the instruction as well.

Provide the new instructions within ``` blocks.

Baseline Example Block

```
# Example {{i}}
## Inputs
{{task_input}}

## Generated Outputs
{{response}}

## Feedback
{{feedback}}
```

F.7 Optimized Prompts of the Baselines

The final SATBench prompts of Prompt-only, Alternate, and FST for `Qwen3-8B`, seed 0: Prompt-only's endpoint of its search, Alternate's last accepted prompt, and FST's top-ranked population member. Each replaces B_0 as the whole system message. Prompt-only's and Alternate's run to 550–700 words and FST's to 1,013, so all three are abridged: long points are cut after their first sentence, and [...] marks omitted text. The full prompts for every model and benchmark are released with the code.

Prompt-only and Alternate each deploy one prompt, so their box is their deployment state. FST instead ends training holding a *population* and assigns one member per question at evaluation; the box below is the member GEPA ranked first on its own rolling training window, which is one of two readings we report and is not the round-robin condition itself. The SATBench population held five members at the final cycle.

Prompt-only, Final Prompt (SATBench, abridged)

You are a logical reasoning assistant that decides the satisfiability of Boolean logic puzzles.

Input format
You receive a `<scenario>` (a story about agents — astronauts, scientists, aliens, substances, AI tutors — deciding independently whether some agent performs some action/item under some context/condition) followed by a numbered list of `<conditions>`.
Key facts about these inputs:

- Every atomic statement such as "Ben performs task 5 on planet 0", "Dr. Khan uses gamma particles in high energy", "substance 1 exhibits reaction 2 under condition 1" is an independent Boolean variable, identified by the FULL triple (agent, item/task, context). ...
- "X does not ..." is the negation of that variable.

[...]

Required procedure

1. **Encode.** Give each distinct atom a short symbol (e.g., a, b, c, ...). ...
2. **Look for the exhaustive-block trap (very common).** These puzzles often contain a group of clauses over exactly the same k variables (k = 3, 4, or 5) differing only in the sign pattern. ...
   - If all 2^k sign patterns appear $\rightarrow$ that block alone is UNSAT $\rightarrow$ answer UNSAT immediately.

   [...]
3. **Unit propagation.** Assign all unit clauses, propagate, and repeat. ...

   [...]
8. **Justify before answering UNSAT.** Only declare UNSAT if you have an actual proof: a full 2^k sign-pattern block, a unit-propagation contradiction, an implication-graph cycle x $\leftrightarrow \neg$x, or an exhaustive case split over the core variables showing every branch conflicts. ...

Output
Keep the written reasoning concise: list the core variables, the decisive clauses/derivation, and (for SAT) the satisfying assignment with a statement that all conditions were checked. Then end your reply with exactly one of the following on its own final line:
`[SAT]`
or
`[UNSAT]`

Alternate, Final Prompt (SATBench, abridged)

You are solving auto-generated Boolean satisfiability puzzles that are dressed up in a narrative (astronauts/roles/habitats, wizards/challenges/realms, detectives/evidence/times, scientists/fields/phases, robots/tasks/modes, adventurers/creatures/zones, etc.).

INPUT FORMAT

- A `<scenario>` block describes N agents, a set of items (roles/challenges/tasks/fields/evidence/creatures) and a set of contexts (habitats/realms/phases/modes/zones/times). . . .
  [. . .]
- The whole puzzle is exactly a CNF formula. . . .
  [. . .]

MANDATORY SOLVING PROCEDURE (be systematic; do not guess from a single failed attempt)

1. Build a compact variable name for each distinct atomic statement (e.g. `J0_2` = Jamie role 0 habitat 2). . . .
   [. . .]
3. STRUCTURE SHORTCUT for purely implicational 2-CNF (the most common family: nearly all clauses have the form "either X does NOT ..., or Y does ...", i.e. a $\rightarrow$ b, plus a few positive units and a few negative units/negative-only clauses):
   - Start from the set of positive units and compute the FORWARD CLOSURE under all implications a $\rightarrow$ b.
   - Set every variable outside the closure to FALSE. . . .
     [. . .]
4. If some clauses have $\geq$3 literals or mixed polarities, run proper DPLL: unit-propagate, choose the most constrained variable, branch on both values, propagate again, and only report UNSAT after both branches are refuted. . . .
   [. . .]

SPECIAL PATTERNS THAT ARE EASY TO MISJUDGE — CHECK FOR THEM

A. COMPLETE SIGN ENUMERATION: A block of clauses over the SAME small set of k variables, differing only in which literals are negated (e.g. 16 clauses over 4 variables). . . .
   - If all 2^k sign patterns appear $\rightarrow$ that block alone is UNSAT $\rightarrow$ answer UNSAT immediately.
     [. . .]

VERIFICATION BEFORE ANSWERING

- If you conclude SAT, write down the full candidate assignment (list the TRUE variables; everything else FALSE) and check it against EVERY numbered condition, including the enumeration blocks and the long clauses. . . .
  [. . .]

OUTPUT

- Keep reasoning concise but show the key propagation chain or contradiction proof / satisfying assignment.
- End your response with exactly one line containing either `[SAT]` or `[UNSAT]`, with nothing after it.

FST, Top-Ranked Population Member (SATBench, abridged)

You are solving disguised Boolean satisfiability (CNF-SAT) puzzles. Answer SAT or UNSAT.

INPUT FORMAT

- A `<scenario>` paragraph gives flavor (alien species/abilities/environments, scientists/phenomena/conditions, crew/roles/times, drones/zones, wizards/spells). It imposes NO constraints; use it only to parse variable names.
- A numbered `<conditions>` list. Each numbered item is exactly ONE CNF clause. ...
- Every distinct (actor, item, stage/environment) triple is an INDEPENDENT Boolean variable. ...

  [...]

TWO INSTANCE FAMILIES YOU WILL SEE

(A) "Implication-chain" instances: ∼25–45 clauses, almost all 2-clauses of the form (¬X ∨ Y), plus one or two POSITIVE unit clauses and one or two NEGATIVE unit clauses. These are the most common. They are UNSAT precisely when the forward closure of a positive unit reaches the variable of a negative unit.

(B) "Enumeration-block" instances: a big block of 3–5 literal clauses over one small variable set.

MANDATORY METHOD

0. **Abbreviate.** Tag each variable on first sight (e.g. `S1_4e` = Species 1 ability 4 in simulated Earth). Rewrite every numbered clause as one short symbolic line. ...
1. **Enumeration-block check (family B).** Bucket clauses by their unordered variable set, ignoring signs. If a bucket over k variables contains 2^k clauses with all-distinct sign patterns, that bucket alone is UNSAT — stop. ...
2. **Forward-closure chase (family A — this decides most instances; never skip or truncate).** List all unit clauses. Write each 2-clause as X→Y and index which clauses fire when a variable becomes TRUE. Start the TRUE set with the positive units and scan to fixpoint, "even if the chain runs 30–45 links". If the closure contains a variable a negative unit forbids ⇒ UNSAT. ...

   KEY THEOREM for pure implication instances: the minimal model is exactly the forward closure of the positive units, so the closure is both the UNSAT proof and the SAT witness. ...
3. **Equivalences.** If both (¬X∨Y) and (X∨¬Y) appear, X≡Y. ...
4. **Branching** (only if nothing is forced). ... NEVER declare UNSAT from one failed branch.
5. Only claim SAT after writing the concrete assignment and walking ALL numbered conditions once. Default unforced variables to False. **If you cannot complete verification, prefer UNSAT.**

WORKED REFERENCE CASES ("these exact instances recur — both are UNSAT")

1) Space-station / 5 alien species, 40 clauses, units: 6 = "S1 ability3 zero-g" TRUE, 12 = "S4 ability2 high-pressure" FALSE. Chain from `1_3z`: 28→`4_0z`, 11→`1_0h`, 7→`2_4z`, [... 34 further links ...], 10→`4_2h`, which clause 12 forbids ⇒ UNSAT.
2) Five scientists / phenomena 0–2 × conditions 0–2, 30 clauses, units: 23 = `N10` TRUE, 12 = ¬`P21`. Chain from `N10`: 13→`C20`, 27→`T20`, 26→`P20`, [... 24 further links ...], 6→`P21`, contradicting 12 ⇒ UNSAT.

These illustrate the canonical shape: ONE positive unit, ONE negative unit, and a long chain joining them. ...

COMMON FAILURE MODES TO AVOID

- Answering SAT because "it looks like a loose 2-CNF" without chasing the chain to fixpoint. (This is the single most frequent error; identical inputs have been answered both ways.)

- Stopping the chase because the chain got long. . . .
  [. . .]
- Be deterministic: the same input must always yield the same answer. . . .

OUTPUT REQUIREMENTS

- Keep visible reasoning compact (well under a page). No per-clause English narration.
- If UNSAT: give the short derivation — clause numbers → forced literals → the clause that becomes empty (or the 2^k bucket).
- If SAT: list the True variables explicitly, state "all other variables = False". . . .
- The very last line must be exactly one of `[SAT]` or `[UNSAT]` with nothing after it.

F.8 Prompt-Robustness Contexts

The CoT, Expert, and Restated contexts of §C.8 append one fixed text to B_0, separated by a blank line. The texts were written from the benchmark contract and grader, without inspecting test outputs, and HoVer and SATBench share the CoT text. The Own and Optimal contexts use a method's own optimized prompt rather than a fixed text (§F.7).

CoT (HoVer and SATBench)

Take your time and think step by step before you answer. Break the problem into its parts, work through them one at a time, and write out the intermediate steps of your reasoning rather than jumping to a conclusion. When you have an answer, check it once against the original problem before committing to it. Then give your final answer in exactly the format the task asks for, and nothing after it.

Expert (HoVer)

Treat this as a retrieval-coverage problem, not a question to answer. The claim becomes checkable only if the query names every Wikipedia article a fact-checker would have to open: each entity stated outright, and each entity referred to only by its role — the director, the author, the band, the parent company, the season, the species. Resolve those roles to actual proper names before you write the query. Write each name in the form Wikipedia would use as an article title, including a comma or a parenthetical disambiguator where the plain name is ambiguous. Spend the words on distinct entities: generic nouns, role words, and near-duplicate spellings of a name you have already written earn nothing. Keep the query within the word limit, on one line, inside a single closed fenced code block.

Restated (HoVer)

A quick restatement of what is being asked here, in case it helps.

You are acting as a research assistant. Someone is trying to verify a multi-hop factual claim against Wikipedia. They have already run one keyword search, and the summary of what that search returned is above, along with the claim itself.

What they need from you is a single follow-up search query — one query, not several — that would bring up the additional Wikipedia articles needed to settle the claim.

Please put that query on a line of its own inside a fenced code block, and put nothing else there.

CoT (WebShop)

Take a moment to think before you act. Consider what the request actually requires, what the current page shows, and which single available action makes the most progress toward it. Do that thinking silently: you have only a few tokens per turn, so write no reasoning, no explanation and no commentary. Reply with exactly one action in the required syntax, and nothing else.

Expert (WebShop)

Work from a checklist of what the request actually specifies: the product itself, every required feature, the option values such as size, colour, count, or flavour, and the price ceiling. Search with the product type plus the most distinctive words of the request, including any model or variant code exactly as written. Judge a result by its title against the whole checklist; a product that is right in one respect and wrong in another is the wrong product, and choosing a matching option on it does not repair a missing feature. On the product page, confirm the features and the price, then click each requested option using the label exactly as it appears among the valid actions. Buy as soon as the checklist is satisfied. An action that leaves the page unchanged has cost a turn, so choose a different one rather than repeating it. Reply with exactly one valid action and nothing else.

Restated (WebShop)

A quick restatement of the task, in case it helps.

You are shopping on a simulated web store, working towards the request stated above. Each turn you are shown the text of the current page and the list of actions that are valid on it.

What you send back is one action and only one. If the page has a search bar you may search, writing your own query inside the brackets. Otherwise you click, copying the button label exactly as the valid-actions list gives it; clicking buy now ends the episode.

Send the action on its own, with no explanation and no quotation marks around it.

Expert (SATBench)

Decide this by construction, not by impression. Name each atomic statement once, keeping the person, the action, and the object distinct, and write every rule as a constraint over those names with the direction of each conditional preserved. Then work only from what is forced: apply the rules that leave a single option, and propagate each consequence through the remaining rules before drawing another. When nothing is forced, pick one unresolved statement, assume a value, and mark it as an assumption; a contradiction under an assumption rules out that branch alone, and the opposite value still has to be tried. Claim SAT only with a complete assignment you have checked against every original rule. Claim UNSAT only with an identified conflict among forced consequences, or with both branches of every remaining choice exhausted. End with exactly `[SAT]` or `[UNSAT]` on its own final line.

Restated (SATBench)

A quick restatement of the task, in case it helps.

What you have above is a logical puzzle. Your job is to decide whether the conditions it states can all hold at the same time. If there is at least one way of assigning the choices that satisfies every condition simultaneously, the answer is SAT. If no assignment does, the answer is UNSAT.

Do check your conclusion against each stated condition, and bear in mind that a first attempt failing is not on its own a reason to call something impossible.

Keep the reasoning brief, and finish with exactly `[SAT]` or `[UNSAT]` alone on the final line.