WHALE: A SIMPLE RECIPE FOR JOINT HARNESS–WEIGHT OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent performance depends jointly on the model parameters and the executable harness code that manages context and control flow. Optimizing either component in isolation can leave the system bottlenecked by its frozen counterpart: weight updates can change which harness is effective, while harness updates can change which model capabilities are exposed. Existing joint-adaptation methods optimize weights and textual prompts but leave the broader harness fixed. We propose **W**eight-**H**arness **A**lternating **LE**arning (**WHALE**), a simple recipe that alternates two phases: updating the model under the current harness, then searching for a better harness under the updated model. We instantiate these two phases with online rejection-sampling fine-tuning and Meta-Harness (Lee et al., 2026), respectively. When to switch is a key design choice: to separate real improvements from noise without over-optimizing against a changing counterpart, WHALE uses either fixed phase durations or an adaptive patience rule over training signals. Using Qwen3.5-2B/4B agents across three domains (search question answering, mathematical reasoning, and chess puzzles), WHALE outperforms weight-only, harness-only, and Fast–Slow Training (Tiwari et al., 2026) by 4.15–24.38 percentage points in best mean@8 accuracy. Either component can be the bottleneck: harness search matches peak weight-only accuracy with far fewer rollouts in SearchQA, but improves math accuracy only after a weight update. Small interleaved updates also outperform stagewise weight-then-harness optimization in accuracy and rollout cost.

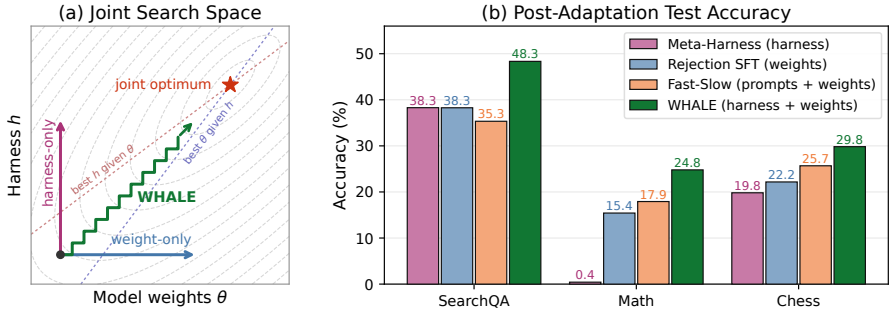

Figure 1: Overview of WHALE. **(a)** Weight-only and harness-only adaptation move along a single axis of $J(\theta, h)$, whereas alternating updates allow both components to co-adapt. **(b)** Our simple joint harness–weight optimization recipe outperforms single-component updates across three diverse domains, as well as joint prompt–weight optimization.

1 INTRODUCTION

Model weights are only half of an agentic language-model system. The other half is the harness, the code that decides which observations enter the context, how tools are exposed and invoked, how execution errors are handled, and when an interaction terminates. Consider a question-answering agent with a retrieval tool: stronger weights cannot use evidence that a brittle harness never retrieves, while better retrieval cannot help a model that cannot synthesize the returned evidence. In this paper,

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.

we aim to address this shifting bottleneck by jointly optimizing the harness and the model for the given problem.

Prior work has jointly adapted model weights and their surrounding context, but has restricted the latter to textual prompts (Soylu et al., 2024; Zhang et al., 2026; Tiwari et al., 2026). Executable tool-using agents expose a broader target for adaptation: the harness code governing tools, observations, error handling, and control flow. Recent LLM-based optimizers make direct search over this code feasible (Hu et al., 2025; Lin et al., 2026; Lee et al., 2026). Although both components can now be optimized directly, how to coordinate model training with full-harness search remains unclear.

Coordinating model training with harness search is difficult because the two phases operate on different timescales. Harness search uses a long proposal-and-accept cycle conditioned on many rollouts, whereas online fine-tuning alternates rollout collection with gradient updates. Running the phases concurrently either evaluates each update against a moving counterpart, thereby confounding credit assignment, or requires synchronization barriers due to their mismatched update cadences. Alternation avoids this tradeoff by decomposing the coupled problem into conditional updates, each performed with respect to a fixed counterpart (Wright, 2015). This raises a key scheduling question: how much should each component change before switching? Each phase must gather enough evidence to distinguish genuine improvements from noise, but stop before over-optimizing against a fixed counterpart.

To study the design decisions surrounding alternating weight–harness updates, we propose **W**eight-**H**arness **A**lternating **LE**arning (**WHALE**), a modular framework that alternates model training under the current harness with harness search under the updated model (Figure 1(a)). In our experiments, we instantiate the weight-update and harness-search phases with online rejection-sampling fine-tuning (RSFT) (Dong et al., 2023; Gulcehre et al., 2023; Shao et al., 2024) and Meta-Harness (Lee et al., 2026), respectively. By default, fixed hyperparameters shared across cycles determine the duration of each phase. We also propose adaptive WHALE, which allows phase durations to vary across cycles by switching, after a minimum duration, when the current phase's training signal stops improving.

We evaluate WHALE against weight-only, harness-only, and Fast–Slow Training (FST, Tiwari et al. (2026)) across three domains: search question answering (SearchQA), mathematical reasoning with code execution (Math), and chess puzzles. To isolate the effect of the harness search space, our FST implementation uses the same update methods and schedule as WHALE but restricts the search to the system and user prompts. WHALE improves over the single-component baselines by 7.67–24.38 percentage points and over FST by 4.15–13.00 points (Figure 1(b)). Across domains, we observe both harness-limited and weight-limited regimes: in SearchQA, harness search alone matches peak weight-only accuracy with far fewer rollouts, whereas in Math it yields almost no improvement until a small weight update makes the same harness search effective. These contrasting regimes show why joint optimization matters: the bottleneck depends on the domain, and updating one component can unlock gains from the other. Compared with stagewise optimization, which spends the full weight-update budget before running the full harness-search budget once, WHALE's small alternating updates improve accuracy by 5.32 and 9.16 percentage points in SearchQA and Math, respectively, and surpass the stagewise result after only 29% and 49% as many rollouts. We also find that adaptive WHALE outperforms the main fixed schedule (with tuned hyperparameters) in both domains we evaluate. Together, our findings provide a controlled, budget-matched study of alternating weight-harness updates and motivate future work on joint weight-harness optimization.

2 Related Work

Training for multi-turn reasoning and tool use. Language-model agents commonly interleave reasoning with actions (Yao et al., 2023), and recent methods train this behavior in search and code-execution environments (Nakano et al., 2022; Chen et al., 2023; Jin et al., 2025; Feng et al., 2026). Prior work on reward-ranked and reinforced self-training methods motivate SFT-style, reward-filtered updates as simpler and more stable alternatives to policy gradient methods (Zelikman et al., 2022; Singh et al., 2024; Dong et al., 2023; Gulcehre et al., 2023; Yuan et al., 2023). We use rejection-sampling fine-tuning from this family for the weight-update phase. Existing methods for training reasoning agents either keep the harness fixed or abstract it away, whereas we study how to train model weights as the harness co-adapts.

Optimizing systems around fixed models. Work outside model weights optimizes increasingly expressive artifacts, from natural-language instructions and modular prompt programs (Yang et al., 2024a; Khattab et al., 2024; Agrawal et al., 2026) to agent architectures and workflows (Hu et al., 2025; Zhang et al., 2025; Zhuge et al., 2024; Shang et al., 2025). Recent methods extend this search to executable harnesses and their components through source-code editing, configuration search, and trajectory-driven updates (Lin et al., 2026; Sengupta & Wang, 2026; Pan et al., 2026; Park et al., 2026). We use Meta-Harness (Lee et al., 2026) as a simple, modular harness-search method that composes directly with weight updates. WHALE connects these lines of work by alternating reward-filtered model training with executable harness search.

Joint and alternating optimization. Alternating optimization methods address coupled problems by updating one component while holding the others fixed (Wright, 2015). Weight and harness adaptation has this structure: each component shapes the trajectories used to improve the other, so fixing one prevents its drift from confounding the active update. Prior methods that jointly adapt prompts and model weights differ in how they coordinate the two updates, but these works restrict the context-side variable to text (Soylu et al., 2024; Ziems et al., 2026; Bo et al., 2026; Lu et al., 2026; Zhang et al., 2026; Tiwari et al., 2026). WHALE instead searches over executable harness programs, motivated by recent work showing that non-prompt harness choices, including tool interfaces, orchestration, memory, and middleware, can substantially alter fixed-model behavior (Yang et al., 2024b; Zhuge et al., 2024; Lee et al., 2026; Lin et al., 2026). Our FST control (Tiwari et al., 2026) uses the same update methods and schedule as WHALE while restricting harness search to the system and user prompts, isolating this difference in expressivity.

3 PRELIMINARIES

Let $\theta \in \Theta$ be the model parameters and $h \in \mathcal{H}$ an executable harness program. The harness includes the system instructions, tool schemas, context management, parser and execution logic, and termination policy. Together, θ and h induce a conditional distribution $\pi_{\theta,h}(\cdot \mid x)$ over trajectories $\tau \in \mathcal{T}$ comprising model messages, tool calls and results, and environment transitions. Given a task distribution $\mathcal{P}$ and verifier $R : \mathcal{T} \to [0, 1]$, we can define the expected reward of the system as:

$$J(\theta, h) \triangleq \mathbb{E}_{x\sim\mathcal{P}}\mathbb{E}_{\tau\sim\pi_{\theta,h}(\cdot|x)}[R(x, \tau)]. \tag{1}$$

We seek a model–harness pair $(\theta^\star, h^\star)$ that maximizes Equation 1. Rather than optimizing this objective directly over $\Theta \times \mathcal{H}$, we separate adaptation into a weight-update phase that updates θ with h fixed and a harness-search phase that updates h with θ fixed. The two remain coupled through $\pi_{\theta,h}$, since each update changes the policy on which the other operates. Many algorithms can instantiate either update; in this work, we use online RSFT and Meta-Harness, respectively, as described below.

3.1 WEIGHT UPDATES: ONLINE REJECTION-SAMPLING FINE-TUNING

We use online rejection-sampling fine-tuning (RSFT) for weight updates: an online variant of rejection-sampling fine-tuning (Shao et al., 2024) that trains the model with supervised learning only on its own verifier-accepted rollouts. Given a fixed harness h, let θ denote the parameters being updated and θ_{old} the frozen copy that generates the rollouts at rollout step s. For a batch $\mathcal{B}_s \subseteq \mathcal{D}_{\text{weight}}$, RSFT samples G trajectories per prompt and keeps only the verifier-accepted pairs: $\mathcal{S}_s^+$ collects every (x, τ) with $x \in \mathcal{B}_s$, $\tau \sim \pi_{\theta_{\text{old}},h}(\cdot \mid x)$, and $R(x, \tau) = 1$. For a trajectory τ, let $\mathcal{I}(\tau)$ denote the positions of model-generated tokens, excluding user prompts and tool results. RSFT maximizes the token-normalized supervised log-likelihood

$$\widehat{J}_{\text{weight}}(\theta; \theta_{\text{old}}, h) = \frac{1}{\sum_{(x,\tau)\in\mathcal{S}_s^+} |\mathcal{I}(\tau)|} \sum_{(x,\tau)\in\mathcal{S}_s^+} \sum_{t\in\mathcal{I}(\tau)} \log p_\theta\left(\tau_t \mid x, h, \tau_{<t}\right). \tag{2}$$

In implementation, RSFT ascends $\widehat{J}_{\text{weight}}$ with minibatch stochastic gradient steps: at rollout step s, it draws SFT minibatches $\mathcal{M} \subseteq \mathcal{S}_s^+$, applies gradient updates, and synchronizes the updated weights to the rollout workers before collecting the next group.

3.2 HARNESS SEARCH: META-HARNESS

We use Meta-Harness (MH) (Lee et al., 2026) for harness search: an iterative proposal–evaluation–selection search over executable harness programs. Given fixed model parameters θ, MH searches for a better harness over I iterations, starting from an initial harness $h^{\text{init}} \in \mathcal{H}$. Let $\mathcal{A}_j$ denote the accumulated harness archive at iteration j, seeded with h^{init}. Evaluating a harness h yields an artifact e_h, comprising its aggregate score, per-example outcomes from the fixed binary verifier R, and saved trajectory logs; $\mathcal{E}_j = \{e_h : h \in \mathcal{A}_j\}$ collects the artifacts of the archived harnesses. The proposer selectively inspects these artifacts and the archived source files through filesystem tools, and one proposer session writes M candidate harnesses $\mathcal{C}_j = \text{Propose}(\mathcal{A}_j, \mathcal{E}_j; M) \subseteq \mathcal{H}$; the archive and its artifacts then grow to $\mathcal{A}_{j+1} = \mathcal{A}_j \cup \mathcal{C}_j$ and $\mathcal{E}_{j+1} = \mathcal{E}_j \cup \{e_h : h \in \mathcal{C}_j\}$. Each candidate $h \in \mathcal{C}_j$ is evaluated on $\mathcal{D}_{\text{harness}}$ with the rollout-based empirical score

$$\widehat{J}_{\text{harness}}(\theta, h) = \frac{1}{|\mathcal{D}_{\text{harness}}|} \sum_{x \in \mathcal{D}_{\text{harness}}} \widehat{\mathbb{E}}_{\tau \sim \pi_{\theta,h}(\cdot|x)}[R(x, \tau)]. \tag{3}$$

Here, $\widehat{\mathbb{E}}$ denotes the empirical mean over the trajectories sampled for each example; our experiments use one trajectory per example during harness search. MH then selects the accepted harness: $h^{\text{accept}} \in \arg\max \widehat{J}_{\text{harness}}(\theta, h)$, where the maximization is over the archive $\mathcal{A}_{j+1}$. After each iteration, MH marks the highest-scoring harness in the expanded archive as accepted. Subsequent proposals can inspect the full archive and its artifacts, allowing them to refine any prior candidate or revert to an earlier design. After I iterations, MH returns the highest-scoring archived harness.

4 WHALE: WEIGHT-HARNESS ALTERNATING LEARNING

Weight-Harness Alternating Learning (WHALE) consists of two alternating phases, each improving one component while holding the other fixed: each cycle trains the model weights under the current harness, then searches for a better harness under the updated model, and repeats. WHALE requires only black-box interfaces for the two update procedures, and can therefore accommodate any weight-update and harness-search method. Starting from an initial model–harness pair (θ_0, h_0), cycle k computes

$$\theta_{k+1} = \text{ModelUpdate}(\theta_k; h_k, \mathcal{D}_{\text{weight}}), \qquad h_{k+1} = \text{HarnessSearch}(h_k; \theta_{k+1}, \mathcal{D}_{\text{harness}}). \tag{4}$$

Budgets and other algorithm-specific settings belong to the instantiation. We instantiate ModelUpdate with RSFT (Section 3.1) and HarnessSearch with MH (Section 3.2). For RSFT, the weight-update phase budget E is the number of training epochs: cycle k consumes the span $\mathcal{D}_{\text{weight}}^{[kE:(k+1)E)}$ from a persistent sampler. For MH, the harness-search phase budget I is the number of harness-search iterations, and M is the number of candidate harnesses proposed per iteration. Algorithm 1 in Appendix A.1 details the WHALE procedure for our instantiation with RSFT and MH.

The two phases use different update data and local objectives, but are coupled through the induced trajectory distribution $\pi_{\theta,h}$: the weight-update phase adapts the model to the prompts, tools, observations, and termination policy defined by h_k, changing behavior such as response length and reasoning patterns, and the harness-search phase then searches for a harness better matched to the updated θ_{k+1}. Repeating the two lets the components co-adapt.

Adaptive WHALE replaces these fixed phase budgets with a per-phase stopping rule. We monitor the training metrics of each phase. After a minimum phase length, the weight-update phase stops once its training reward, averaged over a sliding window of recent steps, has not improved for a fixed number of steps. The harness-search phase stops once the best training score in its archive has not improved for a fixed number of iterations. The rule is early stopping in spirit, but it is applied per phase rather than per run and is driven only by training signals, never by a validation dataset. This removes the schedule (E, I) from the hyperparameters. Algorithm 2 in Appendix A.2 details the procedure and its stopping parameters, and Section 6.2.3 evaluates this adaptive schedule.

5 EXPERIMENTS

Our experiments ask whether alternating weight updates with full-harness search outperforms optimizing either component alone, and whether full-harness search offers gains beyond prompt-only adaptation. We test these questions across three tool-using domains, using shared initializations and matched update budgets within each domain to isolate the effect of joint adaptation.

5.1 DOMAINS AND DATA

SearchQA. Following Search-R1 (Jin et al., 2025), the weight-update training dataset $\mathcal{D}_{\text{weight}}$ contains 18,946 questions: 14,801 from HotpotQA (Yang et al., 2018) and 4,145 from Natural Questions (Kwiatkowski et al., 2019). The harness-search training dataset $\mathcal{D}_{\text{harness}}$ comprises 256 questions, 200 from HotpotQA and 56 from Natural Questions, matching the $\mathcal{D}_{\text{weight}}$ distribution. The test dataset $\mathcal{D}_{\text{test}}$ contains 700 questions, with 100 each from 2WikiMultiHopQA (Ho et al., 2020), Bamboogle (Press et al., 2023), HotpotQA, MuSiQue (Trivedi et al., 2022), Natural Questions, PopQA (Mallen et al., 2023), and TriviaQA (Joshi et al., 2017). Natural Questions, PopQA, and TriviaQA are single-hop, whereas HotpotQA, 2WikiMultiHopQA, MuSiQue, and Bamboogle are multi-hop (Jin et al., 2025). The retrieval environment answers an agent's query with the top-k nearest documents from the Wikipedia 2018 corpus (Karpukhin et al., 2020), indexed using FAISS (Johnson et al., 2021).

Mathematical Reasoning. The weight-update training dataset $\mathcal{D}_{\text{weight}}$ contains 17,917 problems from DAPO-Math-17K (Yu et al., 2025). The harness-search training dataset $\mathcal{D}_{\text{harness}}$ contains 256 problems randomly selected from DAPO-Math-17K. The test dataset $\mathcal{D}_{\text{test}}$ comprises AIME 2024 (Maxwell-Jia, 2024) and AIME 2025 (yentinglin, 2025) problems. Following ReTool (Feng et al., 2026), the agent may write Python programs for intermediate computation; a CPU environment executes them and returns their output or error text as the tool response.

Chess Puzzles. All three datasets are derived from the Lichess open puzzle database (Lichess, 2026). $\mathcal{D}_{\text{weight}}$, $\mathcal{D}_{\text{harness}}$, and $\mathcal{D}_{\text{test}}$ hold 16,384, 256, and another 256 puzzles, mutually disjoint. The agent generates chess moves in Universal Chess Interface (UCI) notation and passes them to an environment that maintains the board state. An incorrect legal move terminates the rollout immediately, whereas an illegal move terminates it after the permitted retry is exhausted. After a correct nonterminal move, the environment returns the puzzle's predefined fixed opponent reply as the tool response. A correct terminal move completes the reference solution sequence and ends the rollout.

Appendix B specifies each domain's harness-search space, initial harness h_0 and binary verifier R. In every domain, the harness search may modify the system and user prompts, the formatting of tool inputs and outputs, the feedback returned after a tool call, and the stopping criteria and turn allocation, while the model, datasets, verifier, environment transition rules, and reference answers stay fixed. The verifier $R(x, \tau) \in \{0, 1\}$ is a sparse trajectory-level reward: intermediate steps receive no reward, and $R(x, \tau) = 1$ only when the terminal task outcome is correct.

5.2 EVALUATION SETUP

SearchQA and Math use Qwen3.5-2B; Chess Puzzles use Qwen3.5-4B (Qwen Team, 2026). Every method starts from the same domain-specific (θ_0, h_0), specified in Appendix B.2. Weight-only holds $h = h_0$ and trains for 4, 6, and 4 epochs in SearchQA, Math, and Chess Puzzles; harness-only holds $\theta = \theta_0$ and runs 40, 60, and 40 harness-search iterations; WHALE alternates the two under $(E, I) = (0.6, 6)$. All rollouts use a sampling temperature of 1.0, top-p of 1.0, and top-k of 20. The response lengths are 8,192, 8,192, and 16,384 tokens for SearchQA, Math, and Chess Puzzles, respectively. The weight update samples $G = 8$ trajectories per prompt, the harness search 1 rollout per candidate–example pair, and test evaluation 8 trajectories per example, reported as mean@8.

Adaptive WHALE keeps this configuration but replaces the fixed budgets with the patience rule of Section 4: the weight-update phase uses a minimum phase length, patience, and averaging window of 0.2 epochs each, and the harness-search phase runs at least 6 iterations and stops after 2 iterations without a new best training score. FST (Tiwari et al., 2026) adapts prompts alongside weights. We

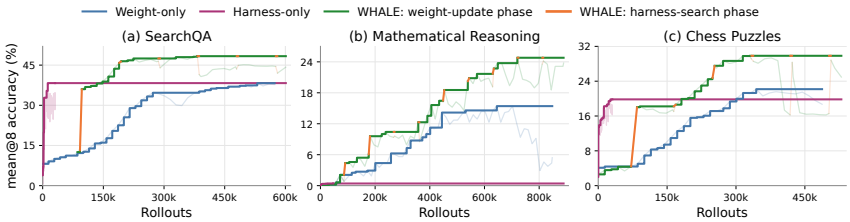

Figure 2: Best-so-far test mean@8 accuracy on the test datasets as rollouts accumulate. Harness-only is held constant after 40, 60, and 40 iterations in SearchQA, Mathematical Reasoning, and Chess Puzzles, respectively.

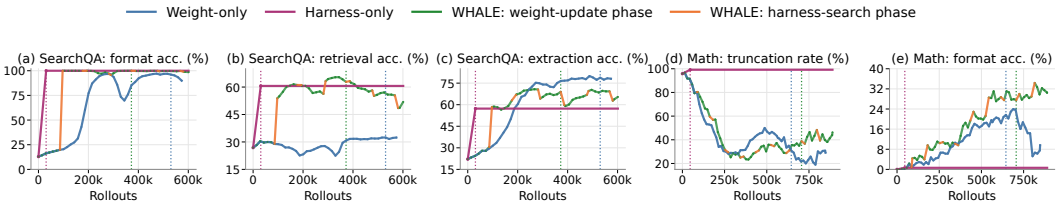

Figure 3: Behavior metrics on test trajectories as rollouts accumulate. **(a–c)** SearchQA: the fraction of responses following the required answer format, the fraction of trajectories retrieving at least one document containing the reference answer, and correctness conditional on retrieving such a document. **(d–e)** Mathematical Reasoning: the fraction of responses reaching the token limit, and the fraction from which the verifier extracts a final boxed answer. Dotted vertical lines mark where each method achieves its best test mean@8 accuracy.

run it as a prompt-restricted control, using the update methods and schedule of WHALE but with harness search confined to the system and user prompts of h_0, thereby isolating search expressivity. The single-component baselines receive budgets matching the cumulative budgets of the WHALE and FST runs; Table 1 in Appendix B shows details for all configurations.

5.3 Does Joint Weight–Harness Optimization Improve Performance?

Figure 2 shows test mean@8 accuracy for single-component baselines and WHALE. Using one fixed schedule, $(E, I) = (0.6, 6)$, WHALE achieves the highest accuracy in all three domains. The ordering of the single-component baselines changes across domains: weight-only and harness-only are effectively tied in SearchQA, whereas weight-only is substantially stronger in Math and Chess Puzzles. WHALE improves under both orderings, outperforming the stronger single-component baseline by 7.67–10.05 percentage points. FST exhibits a different pattern: prompt–weight adaptation underperforms both single-component baselines in SearchQA but outperforms them in Math and Chess Puzzles. WHALE nevertheless exceeds FST by 4.15–13.00 points (Figure 1(b)). Because our FST control uses the same update methods and schedule while restricting harness search to prompts, this comparison measures the benefit of adapting the broader executable harness. The gains over the single-component baselines also hold on every individual benchmark; Table 2 in Appendix C.1 reports the full per-dataset results.

Consistent gains across domains. With one fixed schedule, WHALE outperforms weight-only, harness-only, and prompt-restricted FST across all three domains, despite differences in which component is strongest alone.

6 Analysis

The aggregate gains mask different interactions between model weights and harness code across domains. We first identify which component limits performance and how updating one changes the gains available to the other. We then test whether small interleaved updates improve over stagewise

optimization, how phase duration affects performance, and whether training signals can determine when to switch.

6.1 Which Component Limits Performance?

We observe two domain-dependent bottleneck regimes. SearchQA is *harness-dominant*: harness search matches the peak accuracy of weight updates using only 5.79% as many rollouts. Math is *model-dominant*: harness search reaches only 0.42%, compared with 15.42% for weight updates. These comparisons count target-agent rollouts and exclude proposer compute in the harness search phase.

SearchQA is harness-dominant. Figure 3(a–c) quantifies three capabilities required for SearchQA. Both weight updates and harness search improve format compliance, with weight-only and harness-only peaking at 97.07% and 99.98%: weight updates learn the behavior from format-correct on-policy trajectories, whereas harness search emphasizes the answer format in the user prompt and requires an answer rather than another search on the final turn. Harness-only reaches the higher value with fewer rollouts, so harness search is the more efficient channel here.

For retrieval accuracy, weight-only improves it only modestly, whereas harness-only raises it from 26.88% to 60.61% with far fewer rollouts: the model writes the query, but the harness post-processes it and controls which documents return and how many. Combining both, WHALE reaches the highest retrieval accuracy of 65.41%. Answer extraction reverses the balance, with weight-only and harness-only peaking at 79.93% and 57.28%. In WHALE, the cycle 1 harness-search phase improves extraction sharply, and the cycle 2 weight-update phase improves it further, but from cycle 3 onward the harness-search phases hold it back.

Mathematical reasoning is weight-dominant. Figure 3(d–e) quantifies two capabilities required for Math, where the bottleneck is response truncation. Weight-only cuts the truncation rate from 95.83% to 30.83%, learning to shorten responses from correct on-policy trajectories that terminate before the token limit. Harness-only attempts the same through response caps, turn limits, and a final-answer recovery step, but cannot overcome the base model's behavior, which is also why its format accuracy barely moves. WHALE does shorten responses, reducing truncation and improving format accuracy, and its cycle 1 harness-search phase is the clearest case: in 4,608 rollouts it raises format accuracy from 0.83% to 4.38%, a gain of 3.54 percentage points, where harness-only spends 46,080 rollouts for a gain of 0.63 points (0.00% to 0.63%).

Dominance regimes. We observe both *harness-dominant* and *model-dominant* regimes, and for capabilities that both weight updates and harness search can improve, harness search is more rollout-efficient. A short harness search therefore serves as a low-cost diagnostic before extensive model training. Conversely, a small weight update can make harness search effective where it previously was not.

6.2 How Should Weight and Harness Updates Be Scheduled?

We ablate the alternation schedule (E, I), the per-cycle weight-update epoch and harness-search iteration budgets, in SearchQA and Math, the harness-dominant and model-dominant extremes. Table 3 in Appendix C.2 lists the best point of every run.

6.2.1 Alternating versus Stagewise Optimization

The common way to combine the two is stagewise, with one pass per component: train under h_0 with the full weight-only budget, then run harness search under the resulting model for the full harness-only budget. Each phase selects its best point based on the training score; at the budgets, we denote (E^*, I^*).

In SearchQA, $(E^*, I^*) = (2.7, 32)$ within the 4-epoch and 40-iteration budgets, reaching 43.02% test $\mathrm{mean@8}$ accuracy (Figure 4(a)); in Math, $(E^*, I^*) = (4.3, 33)$ within the 6-epoch and 60-iteration budgets, reaching 15.63% (Figure 4(b)). These fall 5.32 and 9.16 points short of WHALE with $(0.6, 6)$, which passes the final stagewise accuracy after only 29% of the stagewise rollouts in SearchQA and 49% in Math.

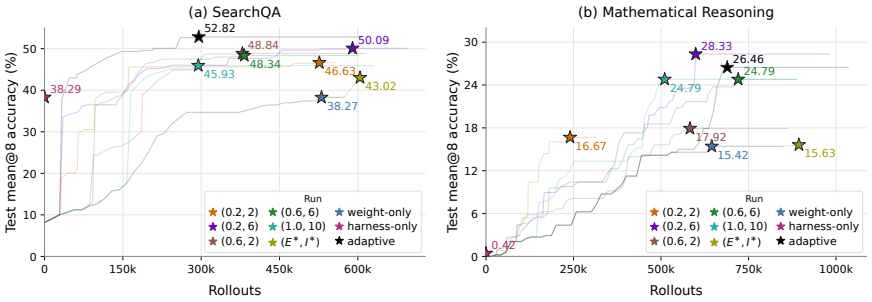

Figure 4: Schedule comparison for (a) SearchQA and (b) Mathematical Reasoning. Each run—the five fixed (E, I) schedules, stagewise (E^*, I^*), the weight-only and harness-only baselines, and adaptive WHALE—is marked by a star at its best test mean@8 accuracy; faint lines trace each run's best-so-far accuracy.

We attribute the weaker stagewise performance to conditional over-optimization (Figure 5). Because $J(\theta, h)$ (Eq. 1) is not separable, each phase optimizes against a counterpart that later changes. A long weight-update phase over-optimizes the model's behavior for h_0, leaving it poorly adapted to the harness produced by the subsequent harness search. A long harness-search phase can likewise over-optimize a harness to $\mathcal{D}_{\text{harness}}$: in Math, the 60-iteration stagewise harness search keeps improving its training score, yet gains only +0.21 percentage points over weight-only on the test set (15.63% versus 15.42%).

Alternation dominates stagewise optimization. One large pass per component is dominated by short alternating phases, in both final accuracy and rollout cost, because spending a component's full budget in a single uninterrupted phase over-optimizes it against a counterpart that then changes.

6.2.2 How Long Should Each Phase Run?

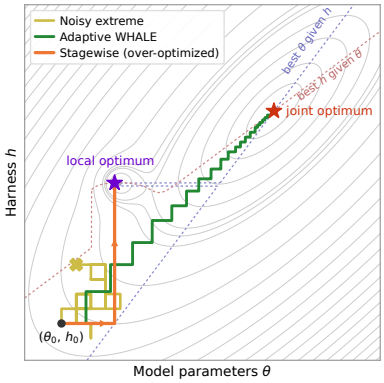

Figure 5: Running each phase to the maximum of its own line ends at a local optimum (the over-optimized extreme); too little evidence per phase stalls on noise (the noisy extreme); adaptive WHALE tracks the valley to the joint optimum.

Given that alternation should take small steps, we next ask how many updates each cycle should contain. In SearchQA and Math, we compare five schedules $(E, I) \in \{(0.2, 2), (0.2, 6), (0.6, 2), (0.6, 6), (1.0, 10)\}$ (Figure 4). Although the two domains lie at opposite extremes of the regime, the same schedule is strongest in both: $(0.2, 6)$ reaches 50.09% in SearchQA and 28.33% in Math, exceeding the $(0.6, 6)$ schedule in the main comparison by +1.75 and +3.54 percentage points, respectively. An effective schedule must therefore lie between two failure extremes: each phase must gather enough evidence to avoid adopting a noisy choice, yet stop before it over-optimizes against its frozen counterpart. The two ends of the schedule spectrum exhibit exactly these failures, sketched in Figure 5.

The noisy extreme is the Math $(0.2, 2)$ run. With only two proposal iterations of evidence per cycle, a chance-inflated candidate is accepted, and the model then adapts to it; the run destabilized and was stopped. At the other end, the over-optimization of Section 6.2.1 sets in well before the stagewise limit: scaling the per-cycle budgets up from $(0.2, 6)$ never helps in either domain, and along $(0.2, 6)$, $(0.6, 6)$, $(1.0, 10)$ accuracy falls from 50.09% to 48.34% to 45.93% in SearchQA and from 28.33% to 24.79% to 24.79% in Math.

Neither budget can be chosen on its own, because the effect of one depends on the value of the other: in Math, increasing E from 0.2 to 0.6 helps at $I = 2$ (16.67% to 17.92%) but hurts at $I = 6$ (28.33%

to 24.79%), and the same crossover holds in the I direction in SearchQA. Even in the model-dominant domain, the schedule that runs harness search most often per training epoch prevails, as explained by catalytic interaction (Section 6.1): each small weight update renews the gains attainable by a bounded harness search.

Effective schedules balance noisy selection and over-optimization. Across both domains, the strongest schedule pairs small weight updates with harness search that runs long enough to identify reliable improvements but stops before over-optimizing.

6.2.3 Can We Automatically Determine When to Switch?

Our sweep over schedule hyperparameters identifies a strong range of phase durations but still requires choosing fixed budgets. We test whether adaptive WHALE can recover this range without fixing (E, I) in advance. For SearchQA and Math, we compare against the main $(0.6, 6)$ schedule and the strongest fixed schedule in our sweep, $(0.2, 6)$. As Figure 4 shows, adaptive WHALE attains the best point of the entire comparison in SearchQA: 52.82% test $\mathrm{mean@8}$ accuracy, +4.48 points over the $(0.6, 6)$ schedule and +2.73 points over the best hand-tuned schedule (50.09%), reached with 23% fewer rollouts than the $(0.6, 6)$ run needs for its own best. In Math, adaptive WHALE reaches 26.46%: +1.67 points over the $(0.6, 6)$ schedule with 4% fewer rollouts, but 1.87 points below the best hand-tuned schedule (28.33%).

The realized schedules fall within the region identified by the ablation (Figure 6). Median phase lengths are 0.24 and 0.29 weight-update epochs in the two domains, with $I = 7$ harness-search iterations per cycle in both, close to the $(0.2, 6)$ optimum, while individual phases extend when their training signal keeps improving, up to 1.16 epochs and $I = 13$.

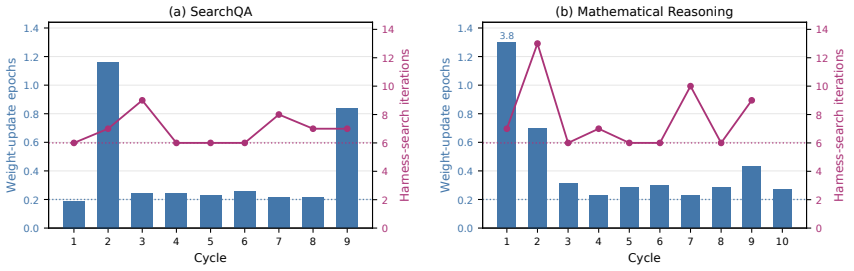

Figure 6: Realized per-cycle budgets chosen by the patience rule of adaptive WHALE in (a) SearchQA and (b) Mathematical Reasoning: weight-update epochs (bars, left axis) and harness-search iterations (dots, right axis). Dotted horizontal lines mark the minimum phase lengths (0.2 epoch, $I = 6$).

7 Conclusion

We presented WHALE, a modular framework that alternates weight updates with harness search over executable agent harnesses. Across SearchQA, Math, and Chess Puzzles it outperforms the weight-only and harness-only baselines by +7.67 to +24.38 percentage points, and FST, whose harness search is restricted to prompts, by +4.15 to +13.00 points. Our analysis shows that domains divide into harness-dominant and model-dominant regimes, and that a small weight update can unlock an otherwise ineffective harness search. A controlled schedule study further shows that one large pass per component is dominated by small alternating steps, and that per-cycle budgets must lie between the noisy and over-optimized extremes. Finally, a per-phase patience rule on training signals automatically identifies this interval, replacing the fixed budgets.

Future work includes instantiating the two operators with other weight-update and harness-search algorithms, as permitted by the black-box interfaces in Section 4, and scaling the study to larger models and additional domains. More broadly, these results motivate treating the model and its harness not as separately engineered artifacts but as one jointly trained system.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods, cleaning and reformatting datasets, interpreting results, and assisting with translation. We have not used them for developing theoretical models or conceptual frameworks, formulating mathematical claims, proposing or refining hypotheses, designing or giving feedback on the methodology or experiments, or supporting qualitative and thematic data analysis; generating synthetic datasets and writing proofs are not applicable to this work. Additionally, we used generative AI tools for creating and editing software code, creating figures, identifying relevant literature, editing the paper for readability, and formatting references. In each case the authors made the decisions and AI tools carried out the implementation or editing: AI coding agents wrote the training code, the Meta-Harness integration, the three environments, and the data-processing, metric-extraction, and plotting scripts, whereas the authors decided which data to filter or split and which metrics to analyze (for example, the five behavior metrics in Figure 3 were chosen by the authors and computed by AI-written code from the raw logs). The authors drafted all text; AI tools were then used for translation and for correcting phrasing, repeated sentences, and inconsistent terminology and notation. References were first compiled by the authors, after which AI tools suggested related work we might have missed. We have reviewed all AI-assisted work: two authors manually reviewed the core AI-written code, all reported numbers were computed from the training logs and saved trajectories, every AI-suggested reference was read and verified by the authors before being cited, and all authors reviewed the AI-edited text in whole or in part, with the first author checking every sentence for false or exaggerated claims. Separately from these aids, LLMs are also components of the systems we study (the Meta-Harness proposer, the SearchQA answer judge, and the Qwen3.5 base models), as described in Section 3.2 and Appendix B. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The complete code is submitted as anonymized supplementary material: the vendored training framework with the online RSFT implementation, the Meta-Harness search implementation and proposer skill for each domain, the three environments with each domain's base harness h_0, the verifiers including the SearchQA judge, data-preparation scripts for the public sources listed in Section 5.1, one launcher per condition (weight-only, harness-only, WHALE, and adaptive WHALE), and the shared alternation loop. Section 5.1 gives the source and size of every dataset split, and Section 5 the sampling and evaluation settings. Algorithms 1 and 2 (Appendix A) state the procedures; Appendix B lists every hyperparameter and budget (Table 1) and the compute used, Appendix B.2 the harness-search space and h_0 of each domain, and Appendix B.3.1 the complete judge prompt. The base models (Qwen3.5-2B and 4B) and the retrieval index are publicly available. Full per-benchmark numbers and the best point of every schedule run are in Tables 2 and 3 (Appendix C), and Appendix D shows complete trajectories. Datasets, trained checkpoints, and harness-search archives are not included in the archive.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/0e9e708b6f48e14fd0ac29e167413f76-Paper-Conference.pdf.

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 C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 116697–116723, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/bd92debabb5e6eb881ef81d88e0f22ae-Paper-Conference.pdf.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. FireAct: Toward language agent fine-tuning, 2023. URL https://arxiv.org/abs/2310.05915.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, KaShun SHUM, and Tong Zhang. RAFT: Reward rAnked FineTuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=m7p5O7zblY.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. ReTool: Reinforcement learning for strategic tool use in LLMs. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 37909–37926, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/4038c9208dfc22644c60ad39c24e5c53-Paper-Conference.pdf.

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

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In Donia Scott, Nuria Bel, and Chengqing Zong (eds.), *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.580. URL https://aclanthology.org/2020.coling-main.580/.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 21344–21377, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/36b7acf6f6010652b3f2a433774a66fe-Paper-Conference.pdf.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021. doi: 10.1109/TBDATA.2019.2921572.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July

2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550/.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54928–54958, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f1cf02ce09757f57c3b93c0db83181e0-Paper-Conference.pdf.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

Yoonho Lee, Roshen Sanjay Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=tmbOUyFx3R.

Lichess. Open database: Puzzle database. https://database.lichess.org/, 2026. Accessed 2026-07-20.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026. URL https://arxiv.org/abs/2604.25850.

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, 2026. URL https://arxiv.org/abs/2603.21877.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9802–9822, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.546. URL https://aclanthology.org/2023.acl-long.546/.

Maxwell-Jia. AIME 2024. Hugging Face dataset, https://huggingface.co/datasets/Maxwell-Jia/AIME_2024, 2024. Accessed 2026-07-20.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. WebGPT: Browser-assisted question-answering with human feedback, 2022. URL https://arxiv.org/abs/2112.09332.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference, 2026. URL https://arxiv.org/abs/2606.05922.

Sungho Park, Wonjoong Kim, Rongyuan Tan, Jue Zhang, Wook-Shin Han, Pengfei Gao, Chanyoung Park, Yongqiang Yao, Rao Fu, Elsie Nallipogu, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. AutoSaddler: Automatic harness optimization with durable updates from agent execution traces, 2026. URL https://arxiv.org/abs/2608.23041.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.378. URL https://aclanthology.org/2023.findings-emnlp.378/.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Biswa Sengupta and Jinhua Wang. HARBOR: Automated harness optimization, 2026. URL https://arxiv.org/abs/2604.20938.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. AgentSquare: Automatic LLM agent search in modular design space. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/0ae94013da7cd459402fd77874e09ee3-Paper-Conference.pdf.

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

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura A Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=lNAyUngGFK. Expert Certification.

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*, pp. 10696–10710, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.597. 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, 2026. URL https://arxiv.org/abs/2605.12484.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training, 2024. URL https://arxiv.org/abs/2212.03533.

Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, Jun 2015. ISSN 1436-4646. doi: 10.1007/s10107-015-0892-3. URL https://doi.org/10.1007/s10107-015-0892-3.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/3339f19c5fcee3ad74502947a32be9e6-Paper-Conference.pdf.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024b. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

yentinglin. AIME 2025. Hugging Face dataset, https://huggingface.co/datasets/yentinglin/aime_2025, 2025. Accessed 2026-07-20.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, juncai liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 113222–113244. Curran Associates, Inc., 2025. doi: 10.52202/085713-3775. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/a4277440d50f1f15d2cb4c14f7e0c0d2-Paper-Conference.pdf.

Hongyi Yuan, Zheng Yuan, Chuanqi Tan, Wei Wang, Songfang Huang, and Fei Huang. RRHF: Rank responses to align language models with human feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 10935–10950. Curran Associates, Inc., 2023. doi: 10.52202/075280-0482. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/23e6f78bdec844a9f7b6c957de2aae91-Paper-Conference.pdf.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488. Curran Associates, Inc., 2022. doi: 10.52202/068431-1126. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/639a9a172c044fbb64175b5fad42e9a5-Paper-Conference.pdf.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, XiongHui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5492ecbce4439401798dcd2c90be94cd-Paper-Conference.pdf.

Lunjun Zhang, Ryan Chen, and Bradly C. Stadie. Evolutionary system prompt learning for reinforcement learning in LLMs, 2026. URL https://arxiv.org/abs/2602.14697.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

Noah Ziems, Dilara Soylu, Lakshya A Agrawal, Isaac Miller, Liheng Lai, Chen Qian, Kaiqiang Song, Meng Jiang, Dan Klein, Matei Zaharia, Karel D'Oosterlinck, Christopher Potts, and Omar Khattab. Composing policy gradients and prompt optimization for language model programs. In *Proceedings of the ACM Conference on AI and Agentic Systems*, CAIS '26, pp. 197–213. ACM, May 2026. doi: 10.1145/3786335.3813164. URL http://dx.doi.org/10.1145/3786335.3813164.

A ALGORITHMS

A.1 WHALE

Algorithm 1 states WHALE for our instantiation with RSFT and MH.

Algorithm 1 WHALE with RSFT and MH

Require: (θ_0, h_0); datasets $\mathcal{D}_{\text{weight}}$, $\mathcal{D}_{\text{harness}}$; fixed binary verifier R; budgets E, I, M

for $k = 0, 1, \ldots$ **do**
  $\theta \leftarrow \theta_k$
  for each prompt batch $\mathcal{B}_s$ in $\mathcal{D}_{\text{weight}}^{[kE:(k+1)E)}$ **do**
    $\theta_{\text{old}} \leftarrow \theta$
    collect $\mathcal{S}_s^+$ under $(\theta_{\text{old}}, h_k)$
    ascend $\widehat{J}_{\text{weight}}(\theta; \theta_{\text{old}}, h_k)$ over minibatches of $\mathcal{S}_s^+$ (Eq. 2)
  end for
  $\theta_{k+1} \leftarrow \theta$; $h^{\text{init}} \leftarrow h_k$; $\mathcal{A}_0 \leftarrow \{h^{\text{init}}\}$; $\mathcal{E}_0 \leftarrow \{e_{h^{\text{init}}}\}$
  for $j = 0, \ldots, I-1$ **do**
    $\mathcal{C}_j \leftarrow \text{Propose}(\mathcal{A}_j, \mathcal{E}_j; M)$
    evaluate $\widehat{J}_{\text{harness}}(\theta_{k+1}, h)$ for each $h \in \mathcal{C}_j$ (Eq. 3), yielding artifacts e_h
    $\mathcal{A}_{j+1} \leftarrow \mathcal{A}_j \cup \mathcal{C}_j$; $\mathcal{E}_{j+1} \leftarrow \mathcal{E}_j \cup \{e_h : h \in \mathcal{C}_j\}$
    $h^{\text{accept}} \leftarrow \arg\max_{h \in \mathcal{A}_{j+1}} \widehat{J}_{\text{harness}}(\theta_{k+1}, h)$
  end for
  $h_{k+1} \leftarrow h^{\text{accept}}$
end for
return the pair (θ_k, h_k) with the best accuracy on $\mathcal{D}_{\text{test}}$

A.2 ADAPTIVE WHALE

Algorithm 2 details adaptive WHALE for our instantiation with RSFT and MH, replacing the fixed budgets E and I of Algorithm 1 with per-phase stopping rules. The weight-update phase uses three parameters, all measured in rollout steps: an averaging window W, a minimum phase length $T_{\min}$, and a patience P_{w}. At step t of a phase, the signal $\bar{r}_t$ is the mean verifier reward of the rollouts collected in the most recent W steps, and $\bar{r}^*$ and t^* track the best signal within the phase and the step that attained it; the phase ends once $t > T_{\min}$ and the signal has not set a new phase best for P_{w} steps. The harness-search phase uses a minimum number of iterations $J_{\min}$, counting the evaluation of the incoming harness as the first, and a patience P_{h}. Here J^* tracks the best training score in the archive and j^* the iteration that attained it; the phase ends once at least $J_{\min}$ iterations have run and J^* has not improved for P_{h} iterations. An improvement of any margin counts. Our experiments set $W = T_{\min} = P_{\text{w}} = 0.2$ epoch and $(J_{\min}, P_{\text{h}}) = (6, 2)$.

B EXPERIMENTAL DETAILS

B.1 PRINCIPAL SETTINGS

Table 1 collects the settings of every reported run. The domains differ only in model size, dataset sizes, total budgets, and context lengths, the longer prompt limit in Chess Puzzles matching its longer board-state observations. Within a domain the four systems draw on the same blocks: weight-only on the first, harness-only on the second, WHALE and FST on both under the per-cycle budgets of the third.

Four points the table does not state. The weight update runs its single SFT epoch over the trajectories accepted in the step that produced them, so no trajectory is revisited later. A harness-search iteration scores every candidate with one rollout per harness-search training example, costing $|\mathcal{D}_{\text{harness}}| \times M$ rollouts in every domain. FST is run with RSFT replacing its reinforcement-learning weight update and MH, restricted to the system and user prompts of h_0, replacing its reflective prompt evolution. The weight-only epoch budgets match the cumulative training epochs of the WHALE and FST runs, and the harness-only iteration budgets their cumulative harness-search iterations.

Algorithm 2 Adaptive WHALE with RSFT and MH

Require: (θ_0, h_0); datasets $\mathcal{D}_{\text{weight}}$, $\mathcal{D}_{\text{harness}}$; fixed binary verifier R; budget M; stopping parameters $(W, T_{\min}, P_{\text{w}})$ and $(J_{\min}, P_{\text{h}})$

for $k = 0, 1, \ldots$ **do**
  $\theta \leftarrow \theta_k$; $t \leftarrow 0$; $\bar{r}^* \leftarrow -\infty$; $t^* \leftarrow 0$
  repeat
    $t \leftarrow t + 1$; $\theta_{\text{old}} \leftarrow \theta$; draw the next prompt batch $\mathcal{B}_t$ from $\mathcal{D}_{\text{weight}}$
    collect $\mathcal{S}_t^+$ under $(\theta_{\text{old}}, h_k)$
    ascend $\widehat{J}_{\text{weight}}(\theta; \theta_{\text{old}}, h_k)$ over minibatches of $\mathcal{S}_t^+$ (Eq. 2)
    $\bar{r}_t \leftarrow$ mean verifier reward of the rollouts of the most recent W steps
    if $\bar{r}_t > \bar{r}^*$ **then**
      $\bar{r}^* \leftarrow \bar{r}_t$; $t^* \leftarrow t$
    end if
  until $t > T_{\min}$ **and** $t - t^* \geq P_{\text{w}}$
  $\theta_{k+1} \leftarrow \theta$; $h^{\text{init}} \leftarrow h_k$; $\mathcal{A}_1 \leftarrow \{h^{\text{init}}\}$; $\mathcal{E}_1 \leftarrow \{e_{h^{\text{init}}}\}$
  $j \leftarrow 1$; $j^* \leftarrow 1$; $J^* \leftarrow \widehat{J}_{\text{harness}}(\theta_{k+1}, h^{\text{init}})$
  repeat
    $j \leftarrow j + 1$; $\mathcal{C}_j \leftarrow \text{Propose}(\mathcal{A}_{j-1}, \mathcal{E}_{j-1}; M)$
    evaluate $\widehat{J}_{\text{harness}}(\theta_{k+1}, h)$ for each $h \in \mathcal{C}_j$ (Eq. 3), yielding artifacts e_h
    $\mathcal{A}_j \leftarrow \mathcal{A}_{j-1} \cup \mathcal{C}_j$; $\mathcal{E}_j \leftarrow \mathcal{E}_{j-1} \cup \{e_h : h \in \mathcal{C}_j\}$
    if $\max_{h \in \mathcal{C}_j} \widehat{J}_{\text{harness}}(\theta_{k+1}, h) > J^*$ **then**
      $J^* \leftarrow \max_{h \in \mathcal{C}_j} \widehat{J}_{\text{harness}}(\theta_{k+1}, h)$; $j^* \leftarrow j$
    end if
  until $j \geq J_{\min}$ **and** $j - j^* \geq P_{\text{h}}$
  $h_{k+1} \leftarrow \arg\max_{h \in \mathcal{A}_j} \widehat{J}_{\text{harness}}(\theta_{k+1}, h)$
end for
return the pair (θ_k, h_k) with the best accuracy on $\mathcal{D}_{\text{test}}$

All runs used NVIDIA H200 GPUs: eight for the Qwen3.5-2B experiments (SearchQA and Math) and sixteen for the Qwen3.5-4B experiments (Chess Puzzles).

B.2 Harness-Search Spaces and Base Harness

Section 5.1 states what the harness search may modify in every domain, and FST is confined to the system and user prompts of h_0. Below are each domain's additions to that space and its h_0.

SearchQA. The harness search may additionally rewrite or expand a model-generated query before retrieval and set retrieval parameters and the ranking of retrieved documents. Each tool call accepts one query, and at most 16 assistant turns are allowed. The initial harness instructs the model to answer factual questions, use search when needed, and place the final answer within `<answer>` tags; it forwards queries unchanged, returns one passage truncated to 200 tokens, and terminates after two assistant turns or a response without a tool call. The retrieval service is a flat FAISS index over `intfloat/e5-base-v2` embeddings (Wang et al., 2024) of the Wikipedia 2018 corpus, as in Search-R1 (Jin et al., 2025).

Mathematical Reasoning. The harness search may additionally change how Python code is extracted and normalized before execution and how standard output and execution errors are surfaced. Each code-interpreter call accepts one program, and at most 16 assistant turns are allowed. The initial harness has no system prompt and requires a final answer in `\boxed{...}`; it extracts the first code block, inserts a print call for a final expression that is not explicitly printed, returns standard output and error text verbatim, and terminates after two assistant turns or a response without a tool call.

Chess Puzzles. The harness search may additionally change how the board observation is presented, how one UCI move is extracted and malformed or ambiguous outputs are handled, how accepted moves and opponent replies are retained or shortened, the retry budgets, and whether the parsed move is validated against the visible legal-move list; it may not generate, substitute, or search for a

Table 1: Principal experimental settings for the three domains.

Setting	SearchQA	Math	Chess Puzzles
Models and datasets			
Base model	Qwen3.5-2B	Qwen3.5-2B	Qwen3.5-4B
Weight-update training examples	18,946	17,917	16,384
Harness-search training examples, $\lvert\mathcal{D}_{\text{harness}}\rvert$	256	256	256
Test examples	700	60	256
Weight-only configuration			
Weight-only epoch budget	4	6	4
Prompt batch, $\lvert\mathcal{B}_s\rvert$	256	256	256
SFT epochs per accepted trajectory set	1	1	1
Learning rate	10^{-7}	10^{-7}	10^{-7}
SFT minibatch, $\lvert\mathcal{M}\rvert$	64	64	64
Rollouts per prompt, G	8	8	8
Harness-only configuration			
Harness-only iterations, I	40	60	40
Proposer		Claude Opus 4.7	
Proposer sessions per harness-search iteration	1	1	1
Candidate harnesses per harness-search iteration, M	3	3	3
Rollouts per candidate–example pair	1	1	1
WHALE and FST configuration			
Train epochs per cycle, E	0.6	0.6	0.6
Search iterations per cycle, I	6	6	6
Harness search space	full harness (WHALE); system and user prompts (FST)		
Completed cycles	6	9	6
Sampling and evaluation			
Test rollouts per example	8	8	8
Sampling temperature	1.0	1.0	1.0
Top-p	1.0	1.0	1.0
Top-k	20	20	20
Prompt token limit	2,048	2,048	4,096
Response length	8,192	8,192	16,384

move. Retry budgets are capped at 10 each and at most 18 assistant turns are allowed. The initial harness asks for exactly one UCI move chosen from the visible position, presents the board and the complete move history without compaction, gives corrective feedback after malformed or illegal output, permits one retry of each kind, and allows nine assistant turns.

B.3 Binary Verifiers

Trajectory acceptance in training, harness-candidate scoring, and test evaluation share one binary verifier $R(x, \tau) \in \{0, 1\}$ per domain, fixed with its reference answers while θ and h change.

SearchQA. The verifier extracts the answer enclosed by `<answer>` tags in the final assistant response, returning $R(x, \tau) = 0$ without a judge call if parsing fails. Otherwise GPT-5.4-mini, at temperature zero and a 256-token cap, judges the extracted answer against the reference answers under a fixed rubric, and $R(x, \tau) = 1$ on a CORRECT judgment. Appendix B.3.1 reproduces the complete judge prompt.

Mathematical Reasoning. Following DAPO's strict boxed-answer verification procedure (Yu et al., 2025), the verifier extracts the final `\boxed{...}` answer. It returns $R(x, \tau) = 1$ if the extracted answer matches the ground-truth answer and $R(x, \tau) = 0$ if parsing fails or the answers differ.

Chess Puzzles. The verifier parses a single UCI move from the model response, checks its legality with `python-chess`, and compares it with the reference move. Failed parsing, failed legality, or

a legal move other than the reference one gives $R(x, \tau) = 0$; only a completed reference sequence gives $R(x, \tau) = 1$.

B.3.1 SEARCHQA LLM-AS-A-JUDGE PROMPT

The SearchQA judge prompt in full. Placeholders are filled per call with the question, the reference answers, and the extracted model answer.

```
You are an impartial judge for a question-answering task.

Mark CORRECT when predicted answer refers to the same entity/value/fact
as any reference. Acceptable:
(1) Surface variation (case, articles, punctuation)
(2) Alias / abbreviation / full-vs-short form ('Pete' = 'Peter')
(3) Added/removed qualifier ('actor X' = 'X')
(4) Less-specific date/place CONTAINED IN reference ('1962' if reference is
    'September 10, 1962'; 'Wellington' if reference is 'Wellington, New Zealand')
(5) Number/unit form ('4' = 'four', '188 acres' = '188 acre')
(6) Different framing of the SAME geographic feature (e.g., 'Gulf of California'
    = 'between Baja California and Sonora' | both describe the same body of water)
(7) Self-correcting answer | if the prediction explores alternatives but commits
    to a final answer that matches the reference, count CORRECT
    (e.g., 'X... actually, Y' where Y matches reference)

Mark INCORRECT for:
(a) Different specific person/place/song/team/company/year
(b) Numeric/date mismatch where neither contains the other
    ('40 miles' vs '35 miles'; '1985' vs '1986')
(c) Opposite or contradictory meaning (winner vs loser; 'no one' vs a named entity)
(d) Different ENTITY TYPE even if related (city is NOT stadium; block is NOT channel;
    episode title is NOT episode number)
(e) Failure to commit to an entity (only explanation/disclaimer)
(f) Listing multiple distinct candidates joined by separators (slashes "/",
    pipes "|", commas, "or", "either...or") | even if one candidate matches
    the reference. A valid answer commits to a single entity; "Beijing /
    Tokyo / Seoul" listing the gold among 2+ alternatives is INCORRECT.
(g) Verbatim passage or document quote | prediction is a multi-sentence
    passage, paragraph, or doc-formatted text (e.g., contains "Doc N (Title:
    ...)" patterns) that contains the reference answer as a substring but
    is not itself a concise answer. A valid answer is the entity/value
    alone, not the surrounding evidence prose.

Question: {question}
Reference answer(s): {gold}
Predicted answer: {prediction}

Respond with exactly one word: CORRECT or INCORRECT.
```

C ADDITIONAL RESULTS

C.1 MAIN COMPARISON: PER-BENCHMARK RESULTS

Table 2 breaks the main comparison of Section 5.3 down to the seven SearchQA datasets, the two AIME datasets, and the Lichess puzzle test set. WHALE is best on every one, so its domain-level gains are not carried by a subset of benchmarks.

Table 2: Test mean@8 accuracy (%) across domains and test dataset subsets. WHALE uses $(E, I) = (0.6, 6)$. Bold denotes the best result in each column.

	SearchQA								Math			Chess Puzzles	
Method	2Wiki	Bamb.	Hotpot	MuSiQue	NQ	PopQA	Trivia	Avg.	AIME24	AIME25	Avg.	Lichess	Avg.
Harness-only	26.38	36.75	38.38	13.88	46.00	40.13	66.50	38.29	0.83	0.00	0.42	19.82	19.82
Weight-only	28.38	31.50	34.88	14.50	51.00	40.88	66.75	38.27	16.25	14.58	15.42	22.17	22.17
FST	22.25	25.00	34.25	13.63	46.88	40.62	64.75	35.34	15.83	20.00	17.92	25.68	25.68
WHALE	**35.75**	**53.00**	**50.25**	**22.13**	**55.50**	**49.25**	**72.50**	**48.34**	**27.08**	**22.50**	**24.79**	**29.83**	**29.83**

C.2 Schedule Comparison: Best Points

Table 3 tabulates the schedule comparison of Section 6.2.2, plotted in Figure 4.

Table 3: Best test mean@8 accuracy (%) of each run in Figure 4 and the rollouts consumed up to that point. Bold denotes the best accuracy in each domain and underlining the second best.

	SearchQA		Math	
Run	mean@8	Rollouts	mean@8	Rollouts
Weight-only	38.27	530,432	15.42	645,120
Harness-only	38.29	30,720	0.42	46,080
Stagewise (E^*, I^*)	43.02	604,160	15.63	893,952
$(E, I) = (0.2, 2)$	46.63	526,336	16.67	240,128
$(E, I) = (0.2, 6)$	<u>50.09</u>	589,824	**28.33**	599,040
$(E, I) = (0.6, 2)$	48.84	378,880	17.92	582,656
$(E, I) = (0.6, 6)$	48.34	382,464	24.79	720,384
$(E, I) = (1.0, 10)$	45.93	294,400	24.79	510,464
Adaptive WHALE	**52.82**	295,424	<u>26.46</u>	689,152

D Trajectory Examples

For each domain we show one test item that the base model under h_0 answers incorrectly and that WHALE with $(E, I) = (0.6, 6)$ answers correctly, with the two trajectories side by side. Message roles are colour-coded: system prompt, user prompt, model response, tool call, tool response, and, in Chess Puzzles, the environment's reply. Long spans are trimmed where marked.

D.1 SearchQA

The question is *Where was the husband of Octavie Coudreau born?*, whose reference answers are `Sonnac` and `Sonnac, Charente-Maritime`. The base pair scores 0/8, WHALE 8/8.

Base model with the base harness h_0 (incorrect). The harness allows two assistant turns and returns one passage. The first retrieval names the husband but not his birthplace; the model searches again, and the rollout ends with no `<answer>` tag.

SYSTEM PROMPT

```
# Tools

You have access to the following functions:

<tools>
{"type": "function", "function": {"name": "search", "description": "Searches the web for
↪  relevant information based on a single query. Returns the top-k passages for that one
↪  query.\n\nMirrors `verl/tools/search_tool.py:SearchTool.execute

[... trimmed ...]

- If there is no function call available, answer the question like normal with your current
↪  knowledge and do not tell the user about function calls
</IMPORTANT>

Answer factual questions using search when needed.
```

USER PROMPT

```
Question: Where was the husband of Octavie Coudreau born?
Use the search tool if needed. Put the final answer only inside <answer>...</answer>.
```

MODEL RESPONSE

```
I need to find information about Octavie Coudreau, specifically her husband's birthplace.
↪  Let me search for this information.
</think>
```

TOOL CALL

```
<tool_call>
<function=search>
<parameter=query_list>
["Octavie Coudreau husband birthplace"]
</parameter>
<parameter=topk>
5
</parameter>
<parameter=max_doc_tokens>
200
</parameter>
</function>
</tool_call>
```

TOOL RESPONSE

```
<tool_response>
{"result": "Doc 1 (Title: \"Octavie Coudreau\")\nOctavie Coudreau Marie Octavie Coudreau
↪  (\"\"née\"\" Renard; 1867{1938) was a French explorer from Anais, Charente, and author
↪  of several books on French Guiana and northern Brazil. In 1899, the Brazilian states of
↪  Pará and Amazonas hired Coudreau to explore and chart the Amazon region. Her husband
↪  was the French explorer and geographer Henri Coudreau.

[... trimmed ...]
```

MODEL RESPONSE

```
<think>
The search result mentions that Octavie Coudreau's husband was Henri Coudreau, but it
↪  doesn't state where Henri Coudreau was born. Let me try a more specific search to find
↪  information about Henri Coudreau's birthplace.
</think>
```

TOOL CALL

```
<tool_call>
<function=search>
<parameter=query_list>
["Henri Coudreau birthplace"]
</parameter>
<parameter=topk>
3
</parameter>
<parameter=max_doc_tokens>
200
</parameter>
</function>
</tool_call>
```

WHALE with $(E, I) = (0.6, 6)$ (correct). Everything that differs on the prompt side is written by the searched harness. Its system prompt grants three search calls and four assistant turns, against the two turns of h_0, and requires at least one retrieval. Its user prompt appends a checklist: the passage must state the answer verbatim, and the answer must be one specific value. Between turns the harness inserts a state line listing the titles retrieved so far. The model searches again when the first passage misses the birthplace, then commits.

SYSTEM PROMPT

```
[... trimmed ...]

s using search when needed. You have a budget of up to three search calls and up to four
↪  assistant turns total. You MUST consult the retriever at least once before your final
↪  answer; do not rely on parametric memory alone. If your first search does not fully
↪  answer the question, use a follow-up search that targets a MORE SPECIFIC missing fact
↪  (a different entity, a linked concept, or a narrower time/place) | do not repeat the
↪  same query. On or before the fourth turn, commit to your final answer inside
↪  <answer>...</answer> (a single name, date, number, or short noun phrase).
```

USER PROMPT

```
Question: Where was the husband of Octavie Coudreau born?

Use the search tool if needed. After each retrieval, before producing <answer>, verify:
  (1) Does a retrieved passage state the specific answer verbatim (a name, date, or number
  ↪  that directly answers the question)? If not, search again with a DIFFERENT phrasing
  ↪  or clue.
  (2) If the question has multiple constraints (e.g. "X who did A and B"), does your
  ↪  candidate answer satisfy EVERY constraint, not just one?
  (3) Your answer must be a single specific value (name, date, number, or short noun
  ↪  phrase) | not a category, list, or explanation.
Put the final answer only inside <answer>...</answer>.
```

MODEL RESPONSE

```
The user is asking a question: "Where was the husband of Octavie Coudreau born?"

I need to find information about Octavie Coudreau and then find out where her husband was
↪  born.

First, I need to find information about Octavie Coudreau. I should search for information
↪  about her to find out who her husband is, and then find out where that husband was
↪  born.
</think>
```

TOOL CALL

```
<tool_call>
<function=search>
<parameter=query_list>
["Octavie Coudreau", "Octavie Coudreau husband", "Octavie Coudreau"]
</parameter>
<parameter=topk>
3
</parameter>
<parameter=max_doc_tokens>
200
</parameter>
</function>
</tool_call>
```

TOOL RESPONSE

```
<tool_response>
{"result": "Doc 1 (Title: \"Octavie Coudreau\")\nOctavie Coudreau Marie Octavie Coudreau
↪  (\"\"née\"\" Renard; 1867{1938) was a French explorer from Anais, Charente, and author
↪  of several books on French Guiana and northern Brazil. In 1899, the Brazilian states of
↪  Pará and Amazonas hi

[... trimmed ...]
```

USER PROMPT

```
Titles retrieved so far: Octavie Coudreau. You have used 1 of 4 assistant turns; up to 2
↪  more searches are possible. If any title in the list above already contains the answer,
↪  commit now inside <answer>...</answer>. Otherwise, your next search MUST target a NEW
↪  entity or attribute not already represented in the list | do NOT re-query a title
↪  above.
```

TOOL CALL

```
<tool_call>
<function=search>
<parameter=query_list>
["Henri Coudreau birth place", "Henri Coudreau birth"]
</parameter>
<parameter=topk>
3
</parameter>
<parameter=max_doc_tokens>
200
</parameter>
</function>
</tool_call>
```

TOOL RESPONSE

```
<tool_response>
{"result": "Doc 1 (Title: \"Henri Coudreau\")\nHenri Coudreau Henri Anatole Coudreau (6 May
↪  1859 Sonnac ( 10 November 1899, State of Pará, Brazil) was a French professor of
↪  history and geography, explorer and geographer of French Guiana and the tributaries of
↪  the Amazon. At the time

[... trimmed ...]
```

MODEL RESPONSE

```
and the search result explicitly states he was born in **Sonnac** (6 May 1859).

<answer>Sonnac</answer>
</think>

The search results identify her husband as **Henri Coudreau** and the text explicitly
↪  states he was born in **Sonnac**.

<answer>Sonnac</answer>
```

D.2 Mathematical Reasoning

The problem is an AIME 2025 counting question with reference answer `279`. The base pair scores 0/8, WHALE 8/8.

Base model with the base harness h_0 (incorrect). The response never calls the code interpreter. It reasons in prose for 20,035 characters, reaches the token limit and is truncated mid-sentence, so the verifier finds no `\boxed{...}` answer to extract.

SYSTEM PROMPT

```
# Tools

You have access to the following functions:

<tools>
{"type": "function", "function": {"name": "code_interpreter", "description": "A tool for
↪  executing code.", "parameters": {"properties": {"code": {"type": "string",
↪  "description": "The code to execute."}}, "required": ["code"], "type": "object"}}}
</tools>

If you choose to call a function ONLY reply in the following format with NO suffix:

<tool_call>
<function=example_function_name>
<parameter=example_parameter_1>
value_1
</parameter>
<parameter=example_parameter_2>
This is the value for the second parameter
that can span
multiple lines
</parameter>
</function>
</tool_call>

<IMPORTANT>
Reminder:
- Function calls MUST follow the specified format: an inner <function=...></function> block
↪  must be nested within <tool_call></tool_call> XML tags
- Required parameters MUST be specified
- You may provide optional reasoning for your function call in natural language BEFORE the
↪  function call, but NOT after
- If there is no function call available, answer the question like normal with your current
↪  knowledge and do not tell the user about function calls
</IMPORTANT>
```

USER PROMPT

```
There are $8!=40320$ eight-digit positive integers that use each of the digits
↪  $1,2,3,4,5,6,7,8$ exactly once. Let $N$ be the number of these integers that are
↪  divisible by 22. Find the difference between $N$ and 2025.
The answer format must be: \boxed{'The final answer goes here.'}
```

MODEL RESPONSE

```
The user wants to find the number of permutations of the digits $\{1, 2, 3, 4, 5, 6, 7,
↪  8\}$ that form an 8-digit integer divisible by 22. Then subtract 2025 from this number.

Let the digits be $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$ in some order.
The digits are distinct and form the sequence $1, 2, 3, 4, 5, 6, 7, 8$.
The number must be divisible by 22.
$22 = 2 \times 11$. So the number must be divisible by 2 and by 11.

Condition 1: Divisible by 2.
This means the last digit (x_8) must be even. The even digits a

[... trimmed ...]

Positions are $1, 2, 3, 4, 5, 6, 7, 8$.
We need to place digits $D_O$ into positions $1, 3, 5, 7$ and $D_E$ into positions $2, 4,
↪  6, 8$.
Wait, the positions are distinct.
So if we have a fixed set of digits for positions $1,3,5,7$, say $\{a, b, c, d\}$, there
↪  are $4!$ ways
```

WHALE with $(E, I) = (0.6, 6)$ (correct). The harness leaves the opening prompt untouched; the system and user messages are byte-identical to the base pair's. What it changes is the turn budget, seven assistant turns against two, and what follows execution: the tool response carries the interpreter's output and then a reminder to return a settled result as `\boxed{...}` rather than as another call. The switch from prose to a single code call is therefore attributable to the weight updates, which is what Section 6.1 means by model-dominant. The response is 3,149 characters and ends in a boxed answer.

SYSTEM PROMPT

```
# Tools

You have access to the following functions:

<tools>
{"type": "function", "function": {"name": "code_interpreter", "description": "A tool for
↪  executing code.", "parameters": {"properties": {"code": {"type": "string",
↪  "description": "The code to execute."}}, "required": ["code"], "type": "object"}}}
</tools>

If you choose to call a function ONLY reply in the following format with NO suffix:

<tool_call>
<function=example_function_name>
<parameter=example_parameter_1>
value_1
</parameter>
<parameter=example_parameter_2>
This is the value for the second parameter
that can span
multiple lines
</parameter>
</function>
</tool_call>

<IMPORTANT>
Reminder:
- Function calls MUST follow the specified format: an inner <function=...></function> block
↪  must be nested within <tool_call></tool_call> XML tags
- Required parameters MUST be specified
- You may provide optional reasoning for your function call in natural language BEFORE the
↪  function call, but NOT after
- If there is no function call available, answer the question like normal with your current
↪  knowledge and do not tell the user about function calls
</IMPORTANT>
```

USER PROMPT

```
There are $8!=40320$ eight-digit positive integers that use each of the digits
↪  $1,2,3,4,5,6,7,8$ exactly once. Let $N$ be the number of these integers that are
↪  divisible by 22. Find the difference between $N$ and 2025.
The answer format must be: \boxed{'The final answer goes here.'}
```

MODEL RESPONSE

```
We have a problem: "There are $8!=40320$ eight-digit positive integers that use each of the
↪  digits $1,2,3,4,5,6,7,8$ exactly once. Let $N$ be the number of these integers that are
↪  divisible by 22. Find the difference between $N$ and 2025.

So we need to count how many 8-digit numbers formed by digits 1-8 (distinct) that are
↪  divisible by 22. Since 22 = 2*11. So number must be divisible by 2 (i.e., even) and by
↪  11.

The digits a

[... trimmed ...]

So just brute force.

We have digits 1-8. 8! = 40320. That's manageable.

Let's brute force.

We can write Python to count.

Then output final answer.

We must output answer as a boxed value.

Okay, let's compute.
</think>
```

TOOL CALL

```
<tool_call>
<function=code_interpreter>
<parameter=code>
import itertools

def count_divisible_by_22():
    digits = [1,2,3,4,5,6,7,8]
    import itertools
    # generate permutations
    import itertools
    digits = [1,2,3,4,5,6,7,8]
    from itertools import permutations
    count=0
    for p in itertools.permutations(digits, 8):
        # form number
        # digits are 1-8 so no leading zero issue
        # convert to int
        n = int("".join(map(str, p)))
        # check 22
        if n % 22 == 0:
            count+=1
    return count

count = count_divisible_by_22()
count
</parameter>
</function>
</tool_call>
```

TOOL RESPONSE

```
<tool_response>
2304

[reminder] If the answer is now clear from this output, your next reply should be the final
↪  answer in \boxed{<answer>} -- no more tool calls are required. Only call
↪  code_interpreter again if you spotted a specific numerical error.
</tool_response>
```

MODEL RESPONSE

```
<think>
We found 2304 numbers. That's the number of 8-digit integers using digits 1-8 exactly once
↪  divisible by 22.

Now we need to find N = 2304.

Find difference between N and 2025.

2304 - 2025 = 279.

So answer = 2304 - 2025 = 279.

Let's double-check logic: we need 8-digit numbers using digits 1-8 exactly once. That's 8!
↪  = 40320 permutations. We counted those divisible by 22. That's correct.

Thus N = 2304.

N - 2025 = 27

[... trimmed ...]

A number is divisible by 22 if it's divisible by 2 and 11.
- Divisible by 2: last digit must be even. Digits available: 1,2,3,4,5,6,7,8 -> even
↪  digits: 2,4,6,8.
- Divisible by 11: alternating sum of digits must be divisible by 11.

We can brute force to find the count N.

\boxed{279}
```

D.3 Chess Puzzles

The puzzle is a mate in two for Black, shown in Figure 7. The base pair scores 0/8, WHALE 8/8.

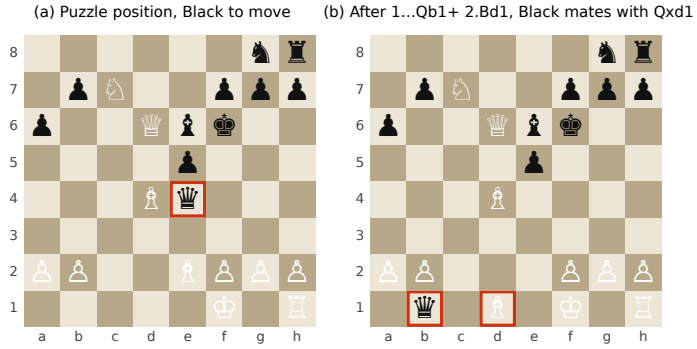

Figure 7: The Chess Puzzles example. **(a)** the puzzle position, with the black queen on e4 outlined. **(b)** after 1 . . .Qb1+ 2.Bd1, where Qxd1 mates.

Base model with the base harness h_0 (incorrect). The base harness states the position as a FEN string and an ASCII board, lists the legal moves flat, and asks for one move. The model never commits: it deliberates to the 8,129-token cap and is cut off mid-word, emitting no `<move>` tag while quoting 21 bracketed UCI tokens from the list, so the verifier reports `__ambiguous_multiple_moves__`. The harness offers a retry, shown below, but the budget is spent and no second turn follows. This is typical: 2,025 of the 2,048 base rollouts stop at the same cap, averaging 8,112 assistant tokens against 764 under WHALE.

SYSTEM PROMPT

```
You solve chess tactics by choosing one legal move at a time.
Use only the board position and legal moves shown in the prompt. Do not claim
hidden engine analysis or external knowledge. Return exactly one move.
```

USER PROMPT

```
Solve the current chess puzzle position.

Task: solve the chess puzzle one move at a time.
Side to move: Black
FEN: 6nr/1pN2ppp/p2Qbk2/4p3/3Bq3/8/PP2BPPP/5K1R b - - 0 19
Board:
8 . . . . . . n r
7 . p N . . p p p
6 p . . Q b k . .
5 . . . . p . . .
4 . . . B q . . .
3 . . . . . . . .
2 P P . . B P P P
1 . . . . . K . R
  a b c d e f g h
Puzzle move number: 1
Legal moves:
- [a6a5] SAN=a5
- [b7b5] SAN=b5
- [b7b6] SAN=b6
- [e4b1] SAN=Qb1+
- [e4c2] SAN=Qc2
- [e4c6] SAN=Qc6
- [e4d3] SAN=Qd3
- [e4d4] SAN=Qxd4
- [e4d5] SAN=Qd5
- [e4e2] SAN=Qxe2+
- [e4e3] SAN=Qe3
- [e4f3] SAN=Qf3
- [e4f4] SAN=Qf4
- [e4f5] SAN=Qf5
- [e4g2] SAN=Qxg2+
- [e4g4] SAN=Qg4
- [e4g6] SAN=Qg6
- [e4h4] SAN=Qh4
- [e5d4] SAN=exd4
- [f6f5] SAN=Kf5
- [f6g5] SAN=Kg5
- [f6g6] SAN=Kg6
- [g7g5] SAN=g5
- [g7g6] SAN=g6
- [g8e7] SAN=Ne7
- [g8h6] SAN=Nh6
- [h7h5] SAN=h5
- [h7h6] SAN=h6

Return exactly one move as <move>uci</move>, for example <move>e2e4</move>.
Do not include explanation.
```

MODEL RESPONSE

```
The user wants me to solve a chess puzzle.
I need to determine the best move for Black given the current position.

**Position Analysis:**
FEN: `6nr/1pN2ppp/p2Qbk2/4p3/3Bq3/8/PP2BPPP/5K1R b - - 0 19`
Side to move: Black

**Board Breakdown:**
- Black pieces:
    - Q on e4
    - N on g8
    - B on b4 (Wait, looking at FEN, 'b' on rank 6 is a king. Wait, let's parse FEN
    ↪  carefully)

[... trimmed ...]

Black Bishop on b7? `1pN2ppp` `p` on b7.
            Black Knight on c7? No.
            Black Bishop on e6? Maybe.
            Is there any protection?
            If `g6h7` pawn exists.
            Wait, `7...`
            Black King on f6.

            Wait, I
```

ENVIRONMENT RESPONSE

```
The verifier could not parse exactly one UCI move from "__ambiguous_multiple_moves__".
↪  Reply with exactly one move, e.g. <move>e2e4</move>, and no explanation.
```

USER PROMPT

```
[... trimmed ...]

[f6g5] SAN=Kg5
- [f6g6] SAN=Kg6
- [g7g5] SAN=g5
- [g7g6] SAN=g6
- [g8e7] SAN=Ne7
- [g8h6] SAN=Nh6
- [h7h5] SAN=h5
- [h7h6] SAN=h6

Feedback from verifier:
The verifier could not parse exactly one UCI move from "__ambiguous_multiple_moves__".
↪  Reply with exactly one move, e.g. <move>e2e4</move>, and no explanation.

Return exactly one move as <move>uci</move>, for example <move>e2e4</move>.
Do not include explanation.
```

WHALE with $(E, I) = (0.6, 6)$ (correct). The prompt below is the searched harness's own work. It parses a piece list from the board and splits it by side, then groups the legal moves by origin square and annotates them `(mate)`, `(check)` and `(capture)`. Each turn restates the moves accepted so far and the opponent's last reply, so the position need not be carried in the reasoning; the environment itself returns only the short line between the two prompts. The harness also fixes how a move is read off a reply: bracketed `[uci]` tokens inside `<think>` are scratch notes, and the last `<move>` tag outside it is the commitment. That rule disposes of the failure above. The model plays Qb1+, sees Bd1 interpose, and mates with Qxd1, in 828 generated tokens.

SYSTEM PROMPT

```
You solve chess tactics one move at a time.
Reply with exactly this two-part structure:
<think>at most three short lines of reasoning about the visible legal moves</think>
<move>PUT_UCI_HERE</move>

Rules for the <move> tag:
- Replace PUT_UCI_HERE with a UCI string copied verbatim from a bracketed
  [uci] entry in the legal moves list, for example <move>e2e4</move> or
  <move>b7f7</move>.
- DO NOT emit the literal p1

[... trimmed ...]

- If you emit multiple <move> tags outside <think> (for example because you
  revised your pick after a second look), the LAST <move> tag is treated as
  your final commitment. Earlier <move> tags are scratch.
Do not claim hidden engine analysis or external knowledge.
```

USER PROMPT

```
Solve the current chess puzzle position.

Side to move: Black
Board:
8 . . . . . . n r
7 . p N . . p p p
6 p . . Q b k . .
5 . . . . p . . .
4 . . . B q . . .
3 . . . . . . . .
2 P P . . B P P P
1 . . . . . K . R
  a b c d e f g h

Piece coordinates (parsed from the ASCII board above, K=king Q=queen R=rook B=bishop
↪  N=knight P=pawn):
Your pieces (you are BLACK, lowercase on the board): Ng8, Rh8, Pb7, Pf7, Pg7, Ph7, Pa6,
↪  Be6, Kf6, Pe5, Qe4
Opponent pieces (WHITE, uppercase on the board): Nc7, Qd6, Bd4, Pa2, Pb2, Be2, Pf2, Pg2,
↪  Ph2, Kf1, Rh1

Legal moves (28 total, grouped by origin square):

From a6 (P on a6) - 1 move(s):
  - [a6a5] SAN=a5

From b7 (P on b7) - 2 move(s):
  - [b7b5] SAN=b5
  - [b7b6] SAN=b6

From e4 (Q on e4) - 15 move(s):
  - [e4b1] SAN=Qb1+  (check)
  - [e4c2] SAN=Qc2
  - [e4c6] SAN=Qc6
  - [e4d3] SAN=Qd3
  - [e4d4] SAN=Qxd4  (capture)
  - [e4d5] SAN=Qd5
  - [e4e2] SAN=Qxe2+  (check, capture)
  - [e4e3] SAN=Qe3
  - [e4f3] SAN=Qf3
  - [e4f4] SAN=Qf4
  - [e4f5] SAN=Qf5
  - [e4g2] SAN=Qxg2+  (check, capture)
  - [e4g4] SAN=Qg4
  - [e4g6] SAN=Qg6
  - [e4h4] SAN=Qh4

From

[... trimmed ...]

- If you cannot fit reasoning, skip <think> and reply only with <move>UCI</move>.
- Any bracketed [uci] tokens inside <think> are treated as scratch notes, not
  as your commitment.
- If you emit multiple <move> tags outside <think> (because you revised your
  pick), the LAST <move> tag outside <think> is treated as your final
  commitment. Earlier <move> tags are scratch.
```

MODEL RESPONSE

```
The position shows Black to move. The legal moves list includes a checkmate move [e4b1]
↪  SAN=Qb1+ which is a check, but I need to verify if it's actually checkmate or just
↪  check. Looking at the legal moves provided, [e4b1] is listed as a legal move for
↪  Black's queen to b1. However, I need to find the best move. The list shows 28 total
↪  legal moves. The most forcing move would be checkmate if available, or check if it l

[... trimmed ...]

</think>

Black's queen can deliver check to the white king on b1.
This is the only check move available in the legal moves list.
<move>e4b1</move>
```

ENVIRONMENT RESPONSE

```
Correct. Opponent replied e2d1 (Bd1). Continue with the next move.
```

USER PROMPT

```
Solve the current chess puzzle position.

Side to move: Black
Board:
8 . . . . . . n r
7 . p N . . p p p
6 p . . Q b k . .
5 . . . . p . . .
4 . . . B . . . .
3 . . . . . . . .
2 P P . . . P P P
1 . q . B . K . R
  a b c d e f g h
Last opponent reply: e2d1
Your accepted moves so far: e4b1

Piece coordinates (parsed from the ASCII board above, K=king Q=queen R=rook B=bishop
↪  N=knight P=pawn):
Your pieces (you are BLACK, lowercase on the board): Ng8, Rh8, Pb7, Pf7, Pg7, Ph7, Pa6,
↪  Be6, Kf6, Pe5, Qb1
Opponent pieces (WHITE, uppercase on the board): Nc7, Qd6, Bd4, Pa2, Pb2, Pf2, Pg2, Ph2,
↪  Bd1, Kf1, Rh1

Legal moves (23 total, grouped by origin square):

From a6 (P on a6) - 1 move(s):
  - [a6a5] SAN=a5

From b1 (Q on b1) - 10 move(s):
  - [b1a1] SAN=Qa1
  - [b1a2] SAN=Qxa2  (capture)
  - [b1b2] SAN=Qxb2  (capture)
  - [b1c1] SAN=Qc1
  - [b1c2] SAN=Qc2
  - [b1d1] SAN=Qxd1#  (mate, capture)
  - [b1d3] SAN=Qd3+  (check)
  - [b1e4] SAN=Qe4
  - [b1f5] SAN=Qf5
  - [b1g6] SAN=Qg6

From b7 (P on b7) - 2 move(s

[... trimmed ...]

pick), the LAST <move> tag outside <think> is treated as your final
  commitment. Earlier <move> tags are scratch.
```

MODEL RESPONSE

```
<think>
The board position is critical. Black to move. The legal moves include [b1d1] which is
↪  Qxd1# (checkmate). This is the immediate winning move for Black if the pieces are set
↪  up as described, but I must verify if this is indeed a legal move for Black.
The list says "From b1 (Q on b1) ... - [b1d1] SAN=Qxd1# (mate, capture)".
Wait, the prompt says "Your pieces (you are BLACK, lowercase on the board)".
But the leg

[... trimmed ...]

Therefore, the move is b1d1.

<move>b1d1</move>
</think>

Black to move and delivers immediate checkmate with the queen on b1 capturing on d1.
This move ends the game immediately.
The legal moves list confirms [b1d1] is mate.

<move>b1d1</move>
```

ENVIRONMENT RESPONSE

```
Correct. Puzzle solved.
```