ENVSCHED: ENVIRONMENT-AWARE SCHEDULING FOR ROLLOUT-EFFICIENT AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

As executable environments for language agents scale, generating and verifying their rollouts becomes increasingly expensive. Under finite rollout budgets, this raises an important question: can data management at the environment level make agentic reinforcement learning (RL) more efficient by organizing, evaluating, and scheduling task instances together with the environments that determine their tools, states, transitions, and verification? To address this question, we introduce ENVSCHED (**Env**ironment-aware **Sched**uling), an online scheduler that learns from rollout history. ENVSCHED assigns scheduling value through two complementary modeling choices: exploitation prioritizes tasks likely to provide informative reward differences, while coverage spreads each batch across underobserved environments and interaction patterns. ENVSCHED organizes environment–task pairs as nodes in a dual-relation graph, with within-environment edges capturing task semantics and shared tool, state, and verifier structure, and cross-environment edges capturing abstract execution and reward topology. By propagating historical rollout statistics along both edge types, it predicts within-task reward variance. These predictions guide a scheduler that allocates each batch to tasks with high predicted reward variance and underobserved environments and tasks. Environment quotas and a similarity penalty further limit concentration and redundancy, yielding informative and diverse batches without additional pilot rollouts. In RL on Qwen3-8B, ENVSCHED achieves an average score of 48.81 across BFCL-v3, τ-Bench, and ACEBench-Agent after 40 training steps, exceeding the highest score attained by random task selection after 120 steps. It also improves downstream performance across model sizes at the same training budget. Statistical analyses and ablation studies highlight the complementary roles of within-task reward differentiation and cross-environment coverage in these improvements.

1 INTRODUCTION

Large language models (LLMs) are increasingly trained to act in stateful tool environments, where completing a user request requires multi-turn interaction, coordinated tool use, and consistent updates to the underlying state (Bai et al., 2026; Li et al., 2026; Ye et al., 2026). For example, an agent rescheduling a reservation must retrieve the relevant records, check availability and domain rules, and update dependent entries in a valid order. Executable environments support this interaction through database-backed states, tool interfaces, transition logic, and task-specific verifiers. Recent systems have made executable environments a central scaling axis for Agentic RL, expanding training pools from tens to thousands of environments (Xu et al., 2026; Song et al., 2026; Wang et al., 2026; Dong et al., 2026). This expansion comes with substantial training costs: industrial reports describe RL over thousands of environments and post-training runs consuming hundreds of thousands of GPU-hours (Liu et al., 2024; Chen et al., 2025a; Zeng et al., 2026; Xiaomi MiMo Team, 2026). Figure 1 (left) summarizes the growing scale of environments and reported RL resources.

As the number of executable environments grows, the cost of generating and verifying RL rollouts grows with them. Researchers (Zhang et al., 2025; Lyu et al., 2026) and industrial technical reports (Zeng et al., 2026; Xiaomi MiMo Team, 2026) increasingly emphasize that training efficiency must be improved through better organization of the training data itself, including environment de-

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

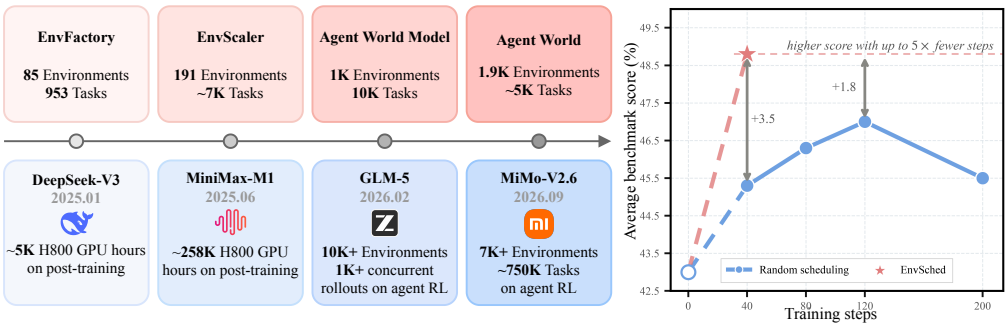


Figure 1: The left panel summarizes the growth of environments and RL resources in recent agent systems. The right panel shows ENVSCHED's training-efficiency gain on Qwen3-8B under matched rollout budgets. Scores are averaged across BFCL-v3, τ-Bench, and ACEBench-Agent.

sign, trajectory generation, rollout scaling, and curriculum learning. Existing work has begun to study online task selection during RL, estimating training value from task difficulty, prompt representations, or historical rewards (Sun et al., 2025; Mao et al., 2026; Liang et al., 2026). However, these methods model the prompt or task while leaving the executable environment outside the selection model. This misses a key source of variation in Agentic RL: the environment determines the tools, initial state, transition rules, and verifier, and thus shapes how a task unfolds and what learning signal it produces. This motivates our formulation of environment-level data management.

We formulate this challenge as *environment-level data management*: organizing, evaluating, and scheduling task instances together with the executable environments in which they are carried out. Each environment specifies the available tools, initial and intermediate states, transition rules, and task verifier. To address this challenge, we draw on the update mechanism of group-relative RL methods such as GRPO (Shao et al., 2024). In these methods, reward differences among rollouts of the same task determine their relative advantages; tasks with nearly identical outcomes therefore provide little learning signal. The scheduler should prioritize tasks whose current reward distributions are likely to yield informative differences. At the same time, repeatedly training on a small set of environments restricts the tools, states, transitions, and verification patterns seen by the policy, even when the selected tasks are individually informative. Effective scheduling must therefore preserve diversity across environments and interaction patterns. Building on these two requirements, we introduce ENVSCHED (**Env**ironment-aware **Sched**uling), an online scheduler learning from rollout history. It prioritizes task instances with high predicted reward informativeness, while it also spreads each batch across underobserved environments and structurally distinct interactions.

ENVSCHED implements exploitation and coverage as two coordinated parts of the same scheduling procedure. To support both decisions, ENVSCHED represents environment–task pairs as nodes in a dual-relation graph. Within-environment edges connect tasks through shared semantics, tools, database fields, and verifier constraints. Cross-environment edges connect tasks through abstract execution and reward topology, enabling information transfer across similar environments. For example, reassigning a consultant and moving a patient between hospital beds can share a dependency structure of entity lookup, eligibility checking, relation replacement, and dependent-state validation.

At each training step, ENVSCHED aggregates historical rollout statistics separately along the two edge types and combines the resulting estimates to predict each task's reward variance under the evolving policy. It then allocates half of each batch to underobserved environments and tasks for coverage, and half to tasks with high predicted reward variance for exploitation. Environment quotas and a similarity penalty limit concentration and redundancy within the shared batch, while newly collected rollout outcomes update the node histories for subsequent rounds. This rollout–update–propagation–selection loop balances policy-dependent task prioritization with explicit environment coverage without additional pilot rollouts.

We evaluate ENVSCHED by conducting reinforcement learning on the EnvScaler training pool with matched per-step rollout budgets. The right side of Figure 1 shows that ENVSCHED reaches an average score of 48.81 on Qwen3-8B across BFCL-v3, τ-Bench, and ACEBench-Agent (Patil et al., 2025; Yao et al., 2025; Chen et al., 2025b) after 40 steps, exceeding the highest score attained by random task selection during 120 steps. It also improves downstream performance at the same

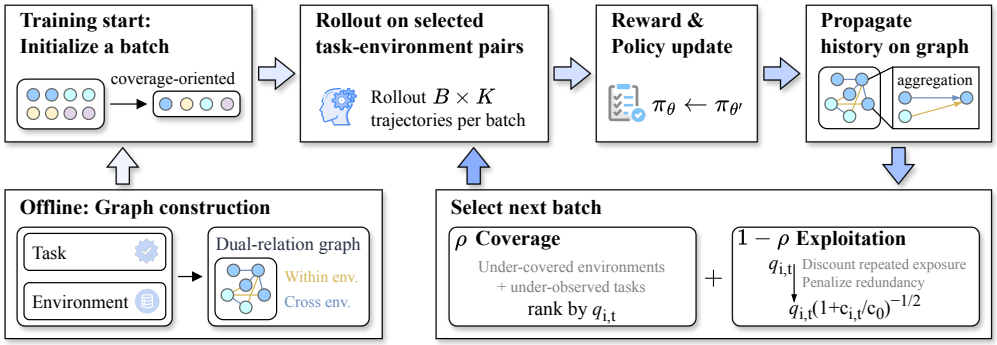


Figure 2: ENVSCHED training loop. Circles with the same color denote task–environment pairs from the same environment.

training budget, with gains across model sizes. Analysis of selected-task rollouts shows that mean within-task reward variance rises from 0.03208 to 0.04290, and environments covered per batch also rises from 36.53 to 45.96 on Qwen3-4B. Ablation studies confirm that reward differentiation and environment coverage are complementary: either component alone is insufficient for effective training. More broadly, ENVSCHED demonstrates the distinctive value to efficient Agentic RL training, paving the way for scaling increasingly capable agent models with better use of training budgets.

Our contributions are threefold:

- We are the first to formulate environment-level data management for Agentic RL, jointly modeling task specifications and executable environment structure.
- We propose ENVSCHED, an Agentic RL training data scheduler which combines a dual-relation graph, historical reward-variance prediction, and coverage-aware batch construction to schedule training tasks without additional pilot rollouts.
- We demonstrate consistent gains in rollout efficiency across models of varying sizes and improved performance on a range of downstream tool-use benchmarks. Our analyses show that reward differentiation and environment coverage are complementary sources of training value, establishing environment-level data management as a promising data-centric direction for scaling efficient Agentic RL.

2 METHOD

2.1 PROBLEM SETUP AND TRAINING LOOP

ENVSCHED treats an environment–task pair as the basic unit of training and scheduling. An environment is an executable setting that defines the tool interfaces, transition rules, state representation, and verifier; a task instance specifies the request and its initial state within that environment. Each pair therefore identifies one concrete rollout problem, and the model interacts with it through text responses and tool calls to produce a trajectory and its task-specific evaluation reward. At each training step, the scheduler chooses a batch of distinct pairs for rollout and model update.

As illustrated in Figure 2, ENVSCHED builds a dual-relation graph from task specifications and executable environment structure before training, then initializes the schedule with a coverage-oriented batch. Rollouts on the selected pairs produce rewards for model updates and statistics for the corresponding node histories. Before the next training step, historical statistics propagate along within-environment and cross-environment edges to estimate task informativeness. The scheduler combines coverage of underobserved environments and tasks with exploitation of informative tasks, using environment quotas and a similarity penalty to limit concentration and redundancy. This rollout–update–propagation–selection loop repeats throughout training.

2.2 DUAL-RELATION GRAPH

We define $\mathcal{G} = (\mathcal{V}, \mathcal{E}^{\text{in}}, \mathcal{E}^{\text{cross}})$ over the environment–task pairs. Each node stores the pair's historical reward statistics and selection count. The two edge types are constructed from similarity:

within-environment similarities compare task semantics and concrete tool, state, and verifier features among pairs in the same environment; cross-environment similarities compare abstract execution and reward structure among pairs in different environments. Similarity determines which observed nodes contribute evidence and how strongly they contribute to the target node's estimate.

Before training, a language model reads the task instruction, initial-state summary, tool descriptions, and verifier to extract a structured operation description. It records required operations, prerequisites, execution dependencies, and relationships between operations and reward checks. For within-environment similarity, six TF–IDF views encode instruction words and character fragments, verifier-check text, verifier code structure, initial context, and the complete operation description. For cross-environment similarity, tool names, field names, entity identifiers, and target values are removed, and operations and conditions are mapped to categories such as lookup, eligibility checking, relation replacement, and dependent-state validation. The LLM prompt is provided in Appendix E. Let $h_i^{(a)}$ encode concrete view a and g_i encode the abstract description. The similarities are

$$S_{ij}^{\text{full}} = \sum_{a=1}^{6} \alpha_a \cos(h_i^{(a)}, h_j^{(a)}), \qquad S_{ij}^{\text{abs}} = \cos(g_i, g_j). \tag{1}$$

The full similarity retains domain-specific information, while the abstract similarity supports transfer between tasks with different business terminology. For example, consultant reassignment and hospital-bed transfer can involve similar lookup, eligibility, and relation-update operations. At each training step, each target node selects its most similar observed neighbors separately within and across environments, using S^{full} and S^{abs}, respectively. We denote these sets by $\mathcal{N}_t^{\text{in}}(i)$ and $\mathcal{N}_t^{\text{cross}}(i)$, with $|\mathcal{N}_t^{\text{in}}(i)| \le K_{\text{within}}$ and $|\mathcal{N}_t^{\text{cross}}(i)| \le K_{\text{cross}}$. Similarities remain fixed during training; node histories and the available observed neighbors evolve. Feature weights and neighbor configurations are specified in Appendix D. Figure 3(a) summarizes this construction: colored regions group task–environment pairs by environment, while the two edge types transfer historical reward information through concrete and abstract structural similarities.

Figure 3(b) summarizes how observed nodes inform estimates for the pool before the next rollout step: the two relation-specific estimates form a graph prior, which is fused with the target's own history and then calibrated into the informativeness score. Each observation records the mean reward m, the within-task reward variance v, and the mean-corrected dispersion d. Let K denote the number of rollouts per task and $r_{i,s,1}, \ldots, r_{i,s,K}$ the rewards of the complete rollout group for task i at step s; we compute the population variance over all K rewards:

$$m_{i,s} = \frac{1}{K}\sum_{k=1}^{K} r_{i,s,k}, \qquad v_{i,s} = \text{Var}_{k=1,\ldots,K}(r_{i,s,k}) = \frac{1}{K}\sum_{k=1}^{K} \left(r_{i,s,k} - m_{i,s}\right)^2. \tag{2}$$

This statistic directly measures the reward differentiation available within the group used for the policy update. The mean reward and mean-corrected dispersion provide complementary information about the same group-level reward distribution; their definitions appear in Appendix B.

2.3 Predicting Task Informativeness

For each statistic $y \in \{v, m, d\}$, we discount older observations and center them against their step-level mean before shifting to the current global reference b_t^y. Let $\bar{y}_{i,t}$ denote the historical estimate and $W_{i,t}$ its effective observation weight. Along each relation $r \in \{\text{in}, \text{cross}\}$, we aggregate observed neighbors with weights $a_{ij,t}^r$ determined by similarity and the amount of recent history:

$$G_{i,t}^{r,y} = \frac{\sum_{j \in \mathcal{N}_t^r(i)} a_{ij,t}^r \bar{y}_{j,t} + \lambda_g b_t^y}{\sum_{j \in \mathcal{N}_t^r(i)} a_{ij,t}^r + \lambda_g}. \tag{3}$$

Shrinkage toward b_t^y stabilizes estimates with little neighbor evidence. We weight the two relation types equally and average their estimates into a graph prior, then combine it with the target's history:

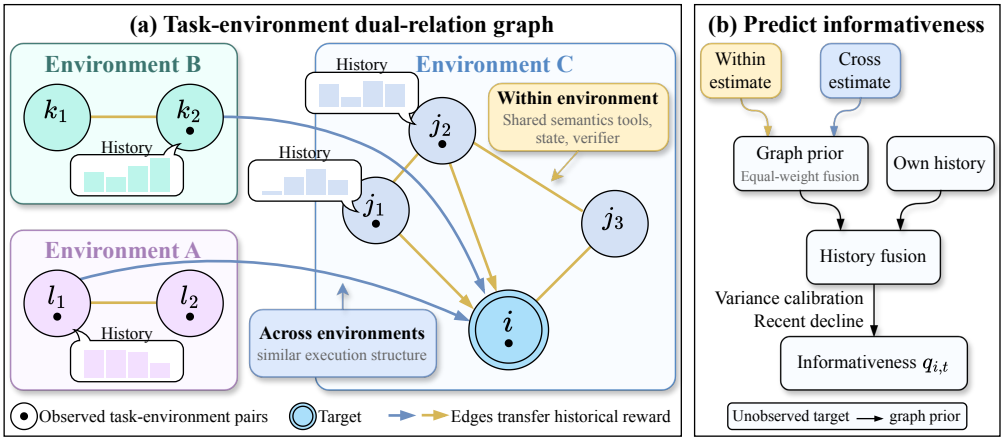


Figure 3: Dual-relation graph and informativeness prediction. (a) Within-environment and cross-environment edges propagate historical statistics from observed task–environment pairs to the target. (b) Relation-specific estimates are fused with the target's history to produce the final score.

$$P^y_{i,t} = \tfrac{1}{2} G^{\text{in},y}_{i,t} + \tfrac{1}{2} G^{\text{cross},y}_{i,t}, \qquad \widehat{y}_{i,t} = \frac{W_{i,t}\bar{y}_{i,t} + \lambda_s P^y_{i,t}}{W_{i,t} + \lambda_s}. \tag{4}$$

An unobserved node relies on the graph prior; its own observations gain weight as its history grows. We combine the predicted variance with a variance estimate reconstructed from the predicted mean and dispersion, then compare estimates from long and short history windows. A recent decline lowers the priority. The resulting percentile score $q_{i,t}$ ranks tasks for scheduling. Appendix B gives the complete calibration, fusion, and scoring definitions.

2.4 Coverage-Aware Task Scheduling

The scheduler assigns a fraction ρ of each batch to coverage and the remaining fraction $1-\rho$ to exploitation. Coverage first restricts the candidate set using a lexicographic priority over current-batch environment coverage, cumulative environment exposure, and task observation count. Within the resulting candidate set, it prioritizes predicted informativeness. Exploitation prioritizes predicted informativeness while gradually discounting repeated exposure. For cumulative selection count $c_{i,t}$, the exposure factor is $\omega_{i,t} = (1 + c_{i,t}/c_0)^{-1/2}$, and the exploitation score is $q_{i,t}\omega_{i,t}$.

Both phases build a shared batch greedily. Let $B_{\text{cov}} = \lfloor \rho B \rfloor$ and $B_{\text{exp}} = B - B_{\text{cov}}$ be the phase budgets. Given the eligible candidates $\mathcal{A}$ and the current selected set $\mathcal{B}$, the next node is

$$i^* = \arg\max_{i \in \mathcal{A}} \left[a^{\text{phase}}_{i,t} - \lambda_{\text{sim}} \max_{j \in \mathcal{B}} S^{\text{full}}_{ij} \right], \tag{5}$$

where the phase score is $q_{i,t}$ for coverage and $q_{i,t}\omega_{i,t}$ for exploitation. For coverage, $\mathcal{A}$ is first restricted by the lexicographic coverage priority described above; Eq. 5 then resolves the remaining choices using informativeness and similarity. The similarity penalty is zero for an empty batch. It uses the full task representation across selected pairs to reduce redundant batch content. Environment quotas and consecutive-selection limits further control concentration.

Candidate eligibility is part of the scheduler and includes two additional filters. A progress filter temporarily excludes tasks with non-positive estimated progress from exploitation while retaining coverage opportunities for reevaluation. An identifier filter uses fixed-action replay to confirm suspicious reward-producing actions and removes the affected tasks from both phases. These filters change candidate eligibility rather than the ranking score. At initialization, the scheduler selects environment representatives and fills the remaining slots using task novelty. The exact eligibility rules and initialization procedure appear in Appendix C.

Table 1: Main results. "Overall" refers to the weighted score. Green and red subscripts denote variations relative to random scheduling.

Model	Steps	BFCL-v3 Multi-Turn					τ-Bench			ACEBench-Agent		
		Base	Miss-Func	Miss-Param	Long-Context	Overall	Retail	Airline	Overall	Multi-Step	Multi-Turn	Overall
Qwen3-1.7B	-	13.50	6.00	12.50	7.00	**9.75**	8.99	16.00	**11.11**	35.00	27.78	**30.67**
+ Random	200	13.17	10.50	10.50	7.67	**10.46**	15.36	16.00	**15.56**	35.00	24.44	**28.67**
+ DPS	200	12.50	10.83	9.33	7.00	**9.92**$_{-0.54}$	16.81	17.33	**16.97**$_{+1.41}$	33.33	26.67	**29.33**$_{+0.66}$
+ LEEPS	200	13.83	11.67	11.00	8.50	**11.25**$_{+0.79}$	16.52	17.33	**16.77**$_{+1.21}$	35.00	27.78	**30.67**$_{+2.00}$
+ EnvSched	40	13.50	13.00	11.50	9.00	**11.75**$_{+1.29}$	17.10	16.67	**16.97**$_{+1.41}$	40.00	24.44	**30.67**$_{+2.00}$
+ EnvSched	200	14.00	14.50	12.17	9.00	**12.42**$_{+1.96}$	18.26	16.00	**17.58**$_{+2.02}$	41.67	24.44	**31.33**$_{+2.66}$
Qwen3-4B	-	32.00	20.00	24.00	25.50	**25.38**	40.87	26.00	**36.36**	56.67	51.11	**53.33**
+ Random	200	36.67	26.17	25.50	27.00	**28.83**	40.87	30.67	**37.78**	70.00	46.67	**56.00**
+ DPS	200	38.33	27.00	26.67	29.50	**30.38**$_{+1.55}$	39.42	30.67	**36.77**$_{-1.01}$	70.00	47.78	**56.67**$_{+0.67}$
+ LEEPS	200	38.00	27.00	26.33	28.67	**30.00**$_{+1.17}$	42.32	30.67	**38.79**$_{+1.01}$	70.00	47.78	**56.67**$_{+0.67}$
+ EnvSched	40	36.00	27.67	26.50	30.50	**30.17**$_{+1.34}$	45.80	30.00	**41.01**$_{+3.23}$	70.00	50.00	**58.00**$_{+2.00}$
+ EnvSched	200	40.67	28.17	27.50	31.17	**31.88**$_{+3.05}$	45.51	30.67	**41.01**$_{+3.23}$	70.00	50.00	**58.00**$_{+2.00}$
Qwen3-8B	-	32.00	33.50	22.00	28.00	**28.88**	46.38	30.00	**41.41**	61.67	56.67	**58.67**
+ Random	200	37.50	34.00	27.50	36.50	**33.88**	56.23	26.67	**47.27**	61.67	51.11	**55.33**
+ DPS	200	40.67	34.17	29.83	37.33	**35.50**$_{+1.62}$	57.68	26.00	**48.08**$_{+0.81}$	65.00	53.33	**58.00**$_{+2.67}$
+ LEEPS	200	41.17	35.33	29.33	37.00	**35.71**$_{+1.83}$	56.23	28.00	**47.68**$_{+0.41}$	60.00	51.11	**54.67**$_{-0.66}$
+ EnvSched	40	45.17	35.83	31.50	38.33	**37.71**$_{+3.83}$	58.26	33.33	**50.71**$_{+3.44}$	70.00	50.00	**58.00**$_{+2.67}$
+ EnvSched	200	46.50	35.00	31.00	39.00	**37.88**$_{+4.00}$	58.26	32.00	**50.30**$_{+3.03}$	70.00	51.11	**58.67**$_{+3.34}$

3 Experiments

3.1 Experimental Setup

Training Environments and Evaluation. We train Qwen3-Thinking (Yang et al., 2025) models with GRPO (Shao et al., 2024) on the EnvScaler (Song et al., 2026) executable-environment pool, which contains 2,550 task instances across 51 executable environments. Each task instance is evaluated through its environment-specific tools, initial state, and terminal verifier. We report performance on BFCL-v3 Multi-Turn, τ-Bench, and ACEBench-Agent (Patil et al., 2025; Yao et al., 2025; Chen et al., 2025b), covering function calling, multi-turn interaction, and long-horizon tool use.

Implementation Details. For training with EnvSched, we apply a unified GRPO configuration across Qwen3-1.7B, Qwen3-4B, and Qwen3-8B. Each training batch contains $B = 64$ environment–task pairs with rollout $K = 8$ per pair (512 trajectories). The scheduler uses a fixed coverage fraction $\rho = 1/2$, yielding 32 coverage pairs and 32 exploitation pairs. We train for 200 single-epoch steps using Adam (lr $= 10^{-6}$, cosine schedule, 10 warmup steps) with a policy-ratio clip of 0.2 and batch-whitened, $[-0.2, 0.2]$-clipped advantages. Rollouts sample at temperature 0.99 (top-$p = 0.99$, top-$k = 100$) with limits of 40 turns, 4,096 tokens per turn, and 32,000 tokens per sequence. The policy loss covers only model-generated tokens, including reasoning; all other tokens are masked. All scheduler and optimization hyperparameters are fixed across model sizes. The default graph neighborhood uses $K_{\text{within}} = 6$ within-environment neighbors and $K_{\text{cross}} = 12$ cross-environment neighbors. Graph representations, scheduling constants, and remaining hyperparameters are listed in Appendix D. For testing on the benchmarks, we use GPT-5.6-terra (Singh et al., 2025) as the user model for τ-Bench, and GPT-4o-2024-08-06 (Hurst et al., 2024) for ACEBench-Agent. We repeat all experiments with three random seeds and report seed-aggregated results.

Baselines. The main comparison includes base model without training, random scheduling, and EnvSched, along with recent related training data scheduling methods DPS (Mao et al., 2026) and LEEPS (Liang et al., 2026). Random samples task–environment pairs uniformly from the training pool. DPS (Dynamics-Predictive Sampling) models prompt-solving dynamics from historical rollout outcomes and selects prompts predicted to provide informative learning signals before rollout generation. LEEPS combines representation-space neighbors, historical rollout outcomes, and explore–exploit portfolios to select prompts likely to yield nonzero reward variance.

3.2 Main Results

We implement all baselines against the same EnvScaler pool and enforce the same batch size, rollout count, training horizon, and model-update procedure as EnvSched. The results on the benchmarks



Figure 4: Selected-task reward variance and environment coverage over training steps, with log-transformed vertical-axis values.

Table 2: Training statistics for random scheduling and ENVSCHED over 200 GRPO training steps. BFCL-v3 Multi-Turn reports the overall score.

	Mean reward variance	Median reward variance	Median reward	Environments covered per batch	Tools used per batch	BFCL-v3 Multi-Turn
Random	0.03208	0.01468	0.595	36.53	413.41	28.83
ENVSCHED	0.04290	0.02108	0.592	45.96	478.39	31.88
Difference	+34%	+44%	-0.5%	+26%	+16%	+3.05

are presented in Table 1. In addition to the results after 200 training steps, we also include the performance of ENVSCHED after just 40 steps. Next, we analyze the results from three aspects:

Consistent gains across model sizes at 200 steps. At the 200-step budget, ENVSCHED improves over random scheduling for all three benchmark overall scores at every model size. Qwen3-4B illustrates the pattern clearly: BFCL-v3 rises from 28.83 to 31.88 (+3.05), τ-Bench from 37.78 to 41.01 (+3.23), and ACEBench-Agent from 56.00 to 58.00 (+2.00). The gains also hold at the ends of the scale: at 1.7B, the improvements are +1.96, +2.02, and +2.66, while at 8B they are +4.00, +3.03, and +3.34 on BFCL-v3, τ-Bench, and ACEBench-Agent, respectively. The BFCL gain at 4B is distributed across all four categories, including Base (36.67 to 40.67), Miss-Function (26.17 to 28.17), Miss-Parameter (25.50 to 27.50), and Long-Context (27.00 to 31.17).

High efficiency. ENVSCHED is already effective after 40 steps within the 200-step training budget. At 4B, it reaches 30.17 on BFCL-v3, 41.01 on τ-Bench, and 58.00 on ACEBench-Agent, matching its 200-step ACEBench-Agent score. At 8B, the 40-step τ-Bench score reaches 50.71, above the 200-step score of 50.30; at 1.7B, ACEBench-Agent reaches 30.67 versus 31.33 after 200 steps. These results show that ENVSCHED can extract useful training signal quickly and, on some benchmarks, match or exceed its longer-run performance.

Prompt-only schedulers do not transfer reliably. DPS and LEEPS are designed around the informativeness of individual prompts, without modeling the environment coupled to each task. Here, choosing a prompt also chooses the tools, state, and constraints exposed to the policy, so prompt-level selection alone cannot reliably identify useful training data. Both methods underperform ENVSCHED overall, and some of their dimensions are no better than random: DPS is below random on 4B τ-Bench (36.77 vs. 37.78), and LEEPS on 8B ACEBench-Agent (54.67 vs. 55.33). The comparison supports combining reward differentiation with environment-aware coverage.

3.3 ANALYSIS

3.3.1 REWARD VARIANCE AND ENVIRONMENT COVERAGE

On Qwen3-4B, ENVSCHED improves both reward differentiation and coverage under the same 64-task batch budget (Table 2). Compared with random scheduling, it increases mean within-task

Table 3: Component ablations and neighborhood-size sensitivity. Panel (a) varies one component of the full scheduler; panel (b) varies one neighbor count at a time in complete 200-step training runs. Benchmark columns report weighted scores. Green and red subscripts denote changes relative to the full ENVSCHED or the default neighborhood size.

Configuration	BFCL-v3 Multi-Turn	τ-Bench	ACEBench Agent	Mean reward variance	Environments per batch	Tools used per batch
Random	28.83	37.78	56.00	0.0321	36.53	413.41
(a) Component contributions						
Full ENVSCHED	31.88	41.01	58.00	0.0429	45.96	478.39
Within-environment edges only	30.42$_{-1.46}$	40.40$_{-0.61}$	58.00	0.0391$_{-0.0038}$	46.70$_{+0.74}$	498.55$_{+20.16}$
Cross-environment edges only	30.63$_{-1.25}$	41.01	57.33$_{-0.67}$	0.0383$_{-0.0046}$	46.18$_{+0.22}$	466.50$_{-11.89}$
Coverage only ($\rho = 1$)	29.71$_{-2.17}$	39.39$_{-1.62}$	56.67$_{-1.33}$	0.0357$_{-0.0072}$	51.00$_{+5.04}$	502.10$_{+23.71}$
Exploitation only ($\rho = 0$)	29.17$_{-2.71}$	38.59$_{-2.42}$	56.00$_{-2.00}$	0.0442$_{+0.0013}$	33.95$_{-12.01}$	420.63$_{-57.76}$
(b) Neighborhood size: $(K_{\text{within}}, K_{\text{cross}})$						
Default $(6, 12)$	31.88	41.01	58.00	0.0429	45.96	478.39
Half within $(3, 12)$	31.25$_{-0.63}$	41.62$_{+0.61}$	56.67$_{-1.33}$	0.0409$_{-0.0020}$	47.18$_{+1.22}$	488.65$_{+10.26}$
Double within $(12, 12)$	31.71$_{-0.17}$	40.81$_{-0.20}$	59.33$_{+1.33}$	0.0434$_{+0.0005}$	45.66$_{-0.30}$	475.71$_{-2.68}$
Half cross $(6, 6)$	30.92$_{-0.96}$	40.20$_{-0.81}$	57.33$_{-0.67}$	0.0425$_{-0.0004}$	45.70$_{-0.26}$	480.01$_{+1.62}$
Double cross $(6, 24)$	32.83$_{+0.95}$	41.01	58.67$_{+0.67}$	0.0441$_{+0.0012}$	46.90$_{+0.94}$	493.83$_{+15.44}$

reward variance from 0.03208 to 0.04290 (+34%) and median variance by 44%, while median reward remains nearly unchanged. Environment coverage rises from 36.53 to 45.96 per batch (+26%) and tool coverage from 413.41 to 478.39 (+16%), alongside a BFCL-v3 improvement from 28.83 to 31.88. The log-transformed curves in Figure 4 show higher variance and coverage for most of the 200 training steps. Over training, variance declines, but the relative magnitude relationships are preserved, while coverage stays stable, preserving reward differentiation and environment diversity.

3.3.2 COMPUTATIONAL COST

ENVSCHED reuses historical training outcomes and requires zero additional probe trajectories. With cached static similarities and fixed-capacity neighbor heaps, an incremental scheduling step costs $O(N\Delta_t \log \hat{K} + N\hat{K} + NB)$, where N is the task-pool size, Δ_t counts newly eligible neighbor candidates, $\hat{K} = K_{\text{within}} + K_{\text{cross}}$ is the neighbor budget, and B is the batch size. For fixed Δ_t, $\hat{K}$, and B, and without boundary-tie fallback scans, this cost is linear in N. In the measured 200-step run, online selection takes 6,985.94 seconds in total, or 34.93 seconds per step. This is a small fraction of the training step: a Qwen3-4B step takes approximately 25 minutes on H100 hardware, so scheduling accounts for about 2.3% of the step time. The incremental backend uses $O(F+N\hat{K}+N)$ persistent space, where F is the number of sparse feature entries. At the 2,550-task scale, sparse features occupy 85.87 MiB, neighbor and progress state occupies approximately 1.24 MiB, and the separately profiled two-matrix representation occupies 42.59 MiB on CPU. These bounded time and memory costs make online environment-aware scheduling practical within the training loop.

3.4 ABLATION STUDIES

3.4.1 COMPONENT CONTRIBUTIONS

We isolate the contributions of the two graph relations and the coverage–exploitation allocation. The relation ablations retain only within-environment edges or only cross-environment edges. The allocation ablations assign all 64 slots to coverage or all 64 slots to exploitation. Table 3 reports performance and statistics. The results show that reward differentiation and environment coverage must be optimized together. Coverage-only scheduling reaches the broadest batches (51.00 environments) but lowers mean reward variance to 0.0357 and degrades all three downstream scores, while exploitation-only scheduling produces the highest variance (0.0442) yet covers only 33.95 environments and performs substantially worse. Combining the two objectives recovers both informative reward differences and broad exposure. Full ENVSCHED has the highest BFCL-v3 score among the component configurations and ties the within-environment-only variant on ACEBench-Agent. The cross-environment-only variant matches the full model on τ-Bench but performs worse on BFCL-v3 and ACEBench-Agent. Both edge ablations also reduce mean reward variance relative to the full

Table 4: Initialization sensitivity over 100 seven-step runs. Panel (a) reports means with 95% confidence intervals and 5th–95th percentile ranges; panel (b) compares the lowest and highest initial-variance quartiles. Hit rate measures exploitation tasks in the highest-variance quartile.

(a) Selection quality across random initializations

Metric	Step 1	Step 7	Step 7 range
Mean batch reward variance	0.0596 [0.0584, 0.0609]	0.0813 [0.0803, 0.0823]	0.0726–0.0894
Cumulative mean reward variance	0.0534 [0.0524, 0.0543]	0.0679 [0.0673, 0.0684]	0.0639–0.0723
Exploitation hit rate (%)	57.2 [55.5, 58.9]	92.1 [91.1, 93.0]	84.4–100.0

(b) Persistence of the initial-batch effect

Step	Mean batch reward variance		Exploitation hit rate (%)	
	Low initial group	High initial group	Low initial group	High initial group
0	0.0406	0.0540	N/A (initial batch)	
1	0.0574	0.0630	55.9	60.4
4	0.0746	0.0768	84.6	83.4
7	0.0810	0.0833	92.8	92.5

graph. Modeling both relations therefore provides effective transfer of rollout evidence across tasks and environments; qualitative examples appear in Appendix F.

3.4.2 Initialization Robustness

We examine sensitivity to the first batch with 100 independent seven-step training runs. Each run starts from a randomly sampled cold-start batch of 64 tasks and then follows the same 32/32 coverage–exploitation scheduler and training procedure. Table 4 reports the mean and confidence interval across runs and compares the bottom and top quartiles ranked by the variance of the initial batch. The experiment measures how much random initialization affects the EnvSched training.

Across the 100 runs, the scheduler quickly moves toward high-signal tasks. Mean batch reward variance rises from 0.0596 at step 1 to 0.0813 at step 7, while the exploitation hit rate increases from 57.2% to 92.1%; even the 5th percentile step-7 branch reaches 84.4%. The effect of the initial batch also decays rapidly. The high-initial-variance group starts 32.8% above the low group, but the gap falls to 9.8% after one step and 2.8% by step 7. At the same point, exploitation hit rates are 92.8% and 92.5% for the low and high groups, respectively. These results show that random cold-start variation affects the first few updates but has little influence on the later task-selection quality.

3.4.3 Neighborhood-Size Sensitivity

We evaluate neighborhood-size sensitivity in complete 200-step training runs by independently halving or doubling either neighbor count around the default setting, $K_{\text{within}} = 6$ and $K_{\text{cross}} = 12$. As shown in Panel (b) of Table 3, performance remains stable across these settings: the largest absolute deviations from the default are only 0.96, 0.81, and 1.33 percentage points on BFCL-v3, τ-Bench, and ACEBench-Agent, respectively. All tested settings outperform random scheduling on all three benchmarks. This stability indicates that EnvSched's gains are robust to the choice of neighborhood size within the tested range, with the default providing a reliable operating point without extensive hyperparameter tuning.

4 Conclusion

We presented EnvSched, an environment-aware scheduler for reinforcement learning with executable agent environments. By treating environment–task pairs as the unit of data management, EnvSched selects tasks with informative reward differentiation while maintaining broad environment coverage. Across Qwen3-1.7B, Qwen3-4B, and Qwen3-8B, it consistently improves performance over random scheduling on BFCL-v3, τ-Bench, and ACEBench-Agent under matched rollout budgets, with gains arising from the complementary effects of reward variance and environment diversity. More broadly, this work first establishes environment-level data management as a useful data-centric perspective for efficient agentic RL, paving the way for scaling increasingly capable agent models with better use of training rollout budgets.

AI USE STATEMENT

As a component of the method. Before training, GPT-5.6-terra reads each task specification and extracts the structured description used to build cross-environment edges in the dual-relation graph. The complete prompt is given in Appendix E.

As part of the evaluation protocol. The τ-Bench and ACEBench-Agent protocols require a user model to simulate multi-turn interaction; the models and versions are reported in Section 3.1.

As a writing aid. LLM assistants were used for grammar, wording, and formatting edits.

LLMs were not used to generate the research idea, to design the scheduling method, or to interpret the experimental results. The authors verified all content and take full responsibility for it.

REPRODUCIBILITY STATEMENT

We describe the full experimental pipeline in the paper and its appendices. The training pool is the publicly released EnvScaler environment pool (Song et al., 2026), from which we use 2,550 task instances across 51 executable environments without modification. Section 3.1 reports the GRPO configuration, optimizer settings, rollout sampling parameters and interaction limits, scheduler batch composition, and graph neighborhood sizes, all of which are held fixed across Qwen3-1.7B, Qwen3-4B, and Qwen3-8B. It also reports the user models and versions used by the τ-Bench and ACEBench-Agent evaluation protocols. All experiments use three random seeds, and we report seed-aggregated results.

The method is specified in full. Appendix B gives the complete calibration, propagation, and fusion equations for historical estimation; Appendix C gives the candidate eligibility rules and the cold-start initialization procedure; Appendix D gives the static representation, the six TF–IDF view weights and encoder settings, and the remaining scheduling constants; and Appendix E gives the complete prompt and abstraction rules used to build cross-environment edges, together with a worked example. Appendix F presents three scheduling decisions with the evidence available at the time, so that individual predictions can be traced end to end.

ETHICS STATEMENT

This work studies reinforcement learning and data selection for language-model agents in tool-use environments. Our experiments run on existing benchmarks and simulated environments, and involve no human subjects and no private or personally identifiable information. Our method may make training autonomous tool-using agents more capable and efficient, but such systems can inherit general safety and misuse risks once deployed in real-world settings. We therefore encourage appropriate safeguards and human oversight in practical deployments.

REFERENCES

Fei Bai, Huatong Song, Shuang Sun, Daixuan Cheng, Yike Yang, Chuan Hao, Renyuan Li, Feng Chang, Yuan Wei, Ran Tao, Bryan Dai, Jian Yang, Wayne Xin Zhao, and Ji-Rong Wen. Clawgym: A scalable framework for building effective claw agents, 2026. URL https://arxiv.org/abs/2604.26904.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025a.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Wang Xinzhi, and Wu Liu. ACEBench: A comprehensive evaluation of LLM tool usage. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 12970–12998, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.697. URL https://aclanthology.org/2025.findings-emnlp.697/.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence, 2026. URL https://arxiv.org/abs/2604.18292.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Woojeong Kim, Ziyi Yang, Jing Nathan Yan, and Jialu Liu. Spend your rollouts where it counts: Rollout allocation for group-based rl post-training, 2026. URL https://arxiv.org/abs/2605.26606.

Jiachun Li, Zhuoran Jin, Tianyi Men, Yupu Hao, Kejian Zhu, Lingshuai Wang, Dongqi Huang, Longxiang Wang, Shengjia Hua, Lu Wang, Jinshan Gao, Hongbang Yuan, Ruilin Xu, Kang Liu, and Jun Zhao. Agentic environment engineering for large language models: A survey of environment modeling, synthesis, evaluation, and application, 2026. URL https://arxiv.org/abs/2606.12191.

Shuang Liang, Haoyang Zhou, Yifan Gong, Guowei Wang, and Xiting Wang. Leeps: Latent-guided explore-exploit prompt sampling for efficient rlvr in large language models, 2026. URL https://arxiv.org/abs/2607.28077.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Zhizhao Liu, Zhiliang Tian, Xi Wang, Zhihua Wen, Yihang Xiong, Zhiquan Lai, and Dongsheng Li. Efficient rlvr scheduling via graph-structured online difficulty estimation, 2026. URL https://arxiv.org/abs/2608.17941.

Yuanjie Lyu, Chengyu Wang, Lei Shen, Jun Huang, and Tong Xu. Mock worlds, real skills: Building small agentic language models with synthetic tasks, simulated environments, and rubric-based rewards. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12529–12545, 2026.

Yixiu Mao, Yun Qu, Qi Wang, Heming Zou, and Xiangyang Ji. Dynamics-predictive sampling for active rl finetuning of large reasoning models. In *International Conference on Learning Representations*, volume 2026, pp. 25242–25270, 2026.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to

agentic evaluation of large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

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

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. EnvScaler: Scaling tool-interactive environments for LLM agent via programmatic synthesis. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.407. URL https://aclanthology.org/2026.findings-acl.407/.

Yifan Sun, Jingyan Shen, Yibin Wang, Tianyu Chen, Zhendong Wang, Mingyuan Zhou, and Huan Zhang. Improving data efficiency for llm reinforcement fine-tuning through difficulty-targeted online data selection and rollout replay. 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. 173449–173476. Curran Associates, Inc., 2025. doi: 10.52202/085713-5770. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/fd54731cc635e2ed6033e8a0e193eddf-Paper-Conference.pdf.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OvLnVkV14P.

Xiaomi MiMo Team. Mimo-v2.6-pro-rl. https://huggingface.co/XiaomiMiMo/MiMo-V2.6-Pro-RL, 2026.

Minrui Xu, Zilin Wang, Mengyi Deng, Zhiwei Li, Zhicheng Yang, Xiao Zhu, Yinhong Liu, Boyu Zhu, Baiyu Huang, Chao Chen, et al. Envfactory: Scaling tool-use agents via executable environments synthesis and robust rl. *arXiv preprint arXiv:2605.18703*, 2026.

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

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for Tool-Agent-User interaction in real-world domains. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 9965–10017, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1b126cc38b8638e07bef37e7b2bb72bf-Paper-Conference.pdf.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. Claw-eval: Towards trustworthy evaluation of autonomous agents, 2026. URL https://arxiv.org/abs/2604.06132.

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.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, et al. The landscape of agentic reinforcement learning for llms: A survey. *arXiv preprint arXiv:2509.02547*, 2025.

A Related Work

A.1 Executable Environments for Agent Training

Tool-use benchmarks provide executable settings for evaluating agents across function calling and multi-turn interaction. BFCL, τ-Bench, and ACEBench assess complementary aspects of tool invocation, user interaction, and task completion (Patil et al., 2025; Yao et al., 2025; Chen et al., 2025b). Recent environment-synthesis systems extend such interaction to diverse training domains. EnvFactory introduces 85 executable environments with 953 task instances for tool-using agents (Xu et al., 2026). EnvScaler (Song et al., 2026) programmatically constructs environment skeletons, initial states, task scenarios, and terminal-state validation functions, producing 191 environments and approximately 7,000 scenarios. Agent World Model (Wang et al., 2026) scales to 1,000 code-driven, database-backed environments with MCP tool interfaces and code-augmented reward verification. Agent-World (Dong et al., 2026) combines environment–task discovery with multi-environment RL and a self-evolving arena that diagnoses capability gaps and synthesizes targeted tasks. These systems expand the supply of executable environments and verifiable tasks for agent training. EnvSched addresses their use during training by organizing environment–task relationships and selecting informative, diverse tasks from the available pool.

A.2 Task Informativeness Estimation and Online Selection

Online task selection adapts the training distribution to the evolving policy by estimating task informativeness from reward variation, historical feedback, and task relationships. Recent technical reports on Agentic RL also treat asynchronous rollout, multi-task mixing, and environment scaling as central data-scheduling problems (Zeng et al., 2026; Xiaomi MiMo Team, 2026). In group-relative RL, rollout reward differences identify informative tasks (Shao et al., 2024); DAPO (Yu et al., 2025) filters uniformly successful or unsuccessful groups and resamples prompts, while Pilot-Commit (Kim et al., 2026) uses reward variation in a pilot sample to allocate later rollouts. For binary rewards, this informativeness is captured by variance $p(1-p)$. DOTS (Sun et al., 2025) selects moderately difficult prompts using reference rollouts and an attention-based difficulty predictor, and Dynamics-Predictive Sampling (DPS) (Mao et al., 2026) models prompt-solving dynamics from historical rollout outcomes to prioritize informative prompts before rollout generation. LEEPS (Liang et al., 2026) combines representation-space neighbors, historical outcomes, and explore–exploit portfolios to identify prompts likely to yield nonzero reward variance without extra rollouts, while graph-structured online difficulty estimation (Liu et al., 2026) propagates feedback over semantic and reasoning similarities through coupled latent difficulty states and a state-level Beta–Binomial model. Building on these foundations, EnvSched incorporates executable environment structure through a dual-relation graph: within-environment edges capture shared tools, database fields, and verifier constraints, while cross-environment edges capture abstract execution and reward topology. These predictions guide task selection with explicit environment coverage, modeling both individual task informativeness and batch-level interaction patterns.

B Complete Historical Estimation

This appendix gives the centering, clipping, and relation-specific neighbor weights used in Section 2.3. For $y \in \{v, m, d\}$, let b_s^y be the mean of y over the tasks observed in training step s. The current global reference and the centered observation are

$$b_t^y = \frac{\sum_{s<t} w_{t,s}^{(h)} b_s^y}{\sum_{s<t} w_{t,s}^{(h)}}, \qquad \widetilde{y}_{i,s} = \mathrm{clip}_{I_y}(y_{i,s} - b_s^y + b_t^y)\,, \tag{6}$$

where $I_v = [0, 0.25]$ and $I_m = I_d = [0, 1]$. If no step-level reference is available, the initialization reference is computed from the first observed batch. The task-history quantities in Eq. 11 use these centered observations.

For the two edge types, $S^{\mathrm{in}} = S^{\mathrm{full}}$ and $S^{\mathrm{cross}} = S^{\mathrm{abs}}$. For relation type q, define the distance between tasks i and j as $\delta_{ij}^q = \mathrm{clip}_{[0,1]}(1 - S_{ij}^q)$. The adaptive bandwidth and neighbor weight are

$$\sigma_{i,t}^{q,(h)} = \max\left\{\text{median}_{j\in\mathcal{N}_t^q(i)}\,\delta_{ij}^q, 0.05\right\}, \qquad a_{ij,t}^{q,(h)} = \exp\left[-\frac{1}{2}\left(\frac{\delta_{ij}^q}{\sigma_{i,t}^{q,(h)}}\right)^2\right]\sqrt{\text{clip}_{[0.1,3]}(W_{j,t}^{(h)})}. \tag{7}$$

When a relation has no available observed neighbor, its estimate is set to the current global reference b_t^y. Percentile ranks are computed over the full task pool of N environment–task pairs and use average ranks for ties:

$$\text{pr}(z_i) = \frac{\text{rank}_{\text{avg}}(z_i) - 0.5}{N}. \tag{8}$$

The empirical quantile function $Q_{\widehat{v}}$ is computed from the predicted variance values of all N tasks and uses linear interpolation. The complete score in Eqs. 12–13 is recomputed independently for the long and short half-lives, including centering, neighbor aggregation, percentile ranks, and quantile mapping.

B.1 Observation Statistics and Priority Construction

For the training implementation, each selected pair produces a complete group of K rewards $r_{i,s,1}, \ldots, r_{i,s,K}$. The overall reward mean, population variance, and mean-corrected dispersion are

$$\begin{aligned} m_{i,s} &= \frac{1}{K}\sum_{k=1}^{K} r_{i,s,k}, & v_{i,s} &= \frac{1}{K}\sum_{k=1}^{K}\left(r_{i,s,k} - m_{i,s}\right)^2, \\ d_{i,s} &= \text{clip}_{[0,1]}\left(\frac{v_{i,s}}{\max\{m_{i,s}(1-m_{i,s}), 0.01\}}\right). \end{aligned} \tag{9}$$

These dispersion and clipping definitions use the training pool's reward scale $[0, 1]$. Historical observations use the half-life weight

$$w_{t,s}^{(h)} = 2^{-(t-s)/h}. \tag{10}$$

For the centered observations in Eq. 6, task histories are summarized as

$$W_{i,t} = \sum_{s\in\mathcal{H}_t(i)} w_{t,s}^{(h)}, \qquad \bar{y}_{i,t} = \frac{\sum_{s\in\mathcal{H}_t(i)} w_{t,s}^{(h)}\widetilde{y}_{i,s}}{W_{i,t}}, \tag{11}$$

Here $\mathcal{H}_t(i)$ contains earlier observation steps. Within a prediction pass using half-life h, $W_{i,t}$ and $\bar{y}_{i,t}$ denote the corresponding $W_{i,t}^{(h)}$ and $\bar{y}_{i,t}^{(h)}$. When no history is available, $W_{i,t} = 0$ and $\bar{y}_{i,t} = b_t^y$. Predicted mean and dispersion reconstruct a second variance estimate, which is combined with the direct variance prediction:

$$f_{i,t} = \widehat{d}_{i,t}\widehat{m}_{i,t}(1-\widehat{m}_{i,t}), \qquad u_{i,t}^{(h)} = Q_{\widehat{v}}[0.65\,\text{pr}(\widehat{v}_{i,t}) + 0.35\,\text{pr}(f_{i,t})]. \tag{12}$$

All predicted quantities and the quantile function in this equation are computed independently for each half-life h. The final priority incorporates recent decline:

$$u_{i,t} = u_{i,t}^{(h_{\text{long}})} - \eta_{\text{decline}}\max\left\{u_{i,t}^{(h_{\text{long}})} - u_{i,t}^{(h_{\text{short}})}, 0\right\}, \qquad q_{i,t} = \text{pr}(u_{i,t}). \tag{13}$$

C Scheduling Rules and Cold Start

The coverage phase first restricts the eligible candidates lexicographically by current-step coverage count, cumulative environment selections, and task observation count; it then applies Eq. 5 using the informativeness score and batch similarity penalty. The hard constraints are parameterized by a per-round environment cap C_1, a rolling three-step environment cap C_3, and a consecutive-task cap C_{task}, set to $C_1 = 4$, $C_3 = 11$, and $C_{\text{task}} = 2$ in our experiments.

The progress filter is part of candidate eligibility. It uses adjacent observations of the same task as training transitions. A ridge predictor estimates the later mean reward from the earlier mean reward, squared mean reward, variance, training step, observation count, and environment category. Each transition is predicted using only transitions from earlier completed training steps. The task's residual history is combined with environment and relation-neighbor references. A task with at least three scorable transitions and a non-positive estimated residual is removed from the exploitation candidates. It remains eligible for coverage and can return to exploitation after new observations change its estimate.

The identifier filter is also part of candidate eligibility. It examines high-reward trajectories for identifiers that cannot be obtained from the permitted context or for replacement objects that duplicate existing records. It first replays the saved fixed action sequence and checks that the replayed reward agrees with the stored reward within 10^{-4}. It then removes the suspicious action and dependent edits and replays the sequence again. A task is filtered only when the removal lowers the reward. Replay uses the saved actions and does not sample a new model trajectory.

At cold start, EnvSched selects one representative task from each environment using the highest mean within-environment similarity. It fills the remaining slots by combining global novelty and within-environment novelty, with no more than two tasks from each environment in the first batch. This initialization uses static structure only.

D Static Representation and Runtime Configuration

The task pool contains 2,550 task instances from 51 environments. The default batch contains $B = 64$ environment–task pairs, with eight rollouts per pair. We use a fixed coverage fraction $\rho = 0.5$, giving 32 coverage slots and 32 exploitation slots. The within-environment and cross-environment neighbor sizes are 6 and 12, respectively; the two relation estimates use fixed equal fusion weights of 0.5 each, and the graph estimate and task-history prior use shrinkage strengths $\lambda_g = \lambda_s = 2$. The long and short half-lives are $h_{\text{long}} = 8$ and $h_{\text{short}} = 2$ training steps, the recent-decline coefficient is $\eta_{\text{decline}} = 0.5$, the exposure-count scale is $c_0 = 16$, and the batch similarity penalty is $\lambda_{\text{sim}} = 0.08$. Environment caps are $C_1 = 4$, $C_3 = 11$, and $C_{\text{task}} = 2$. These hyperparameters are fixed across model sizes.

The six view weights are $(0.28, 0.08, 0.16, 0.12, 0.16, 0.20)$. The six static views use word unigrams and bigrams, character 3–5-grams, verifier-check text, verifier code structure, initial task context, and the complete operation description. Each view has its own TF–IDF encoder with minimum document frequency 1, a 35,000-feature cap, and sublinear term frequency. The structured operation description is extracted offline and normalized for the cross-environment representation. Numeric values are normalized into abstract categories so that cross-environment similarity reflects operation and condition types instead of exact identifiers.

E Cross-Environment Graph Construction Prompt

This appendix provides the complete prompt used to construct cross-environment edges in the dual-relation graph (Section 2.2). Within-environment edges use TF–IDF over concrete features like tool names and verifier code. Cross-environment edges require an abstract representation that removes domain-specific identifiers.

E.1 What This Prompt Extracts

Given a task specification (instruction, initial state, verifier checks, available tools), the prompt outputs a structured description of what operations the task requires and how they depend on each other. After removing domain-specific identifiers, this structure becomes the cross-environment similarity S^{abs} in Eq. 1.

The prompt outputs five components:

1. **Operations**: The atomic actions required to complete the task. Each operation is labeled with a type that describes its effect:
   - `lookup`: retrieve data from the environment (e.g., "find user record by username")
   - `update`: modify existing records (e.g., "change user status field to active")
   - `create`: add new entities to the environment (e.g., "create new reservation")
   - `delete`: remove records
   - `validate`: verify constraints without changing state
   - `authorize`: check permissions

   Each operation lists which other operations must complete before it can start. For example, "update_user_status depends on lookup_user because the user ID from lookup is needed as input to update."
2. **Gates**: Logical conditions that must be satisfied before operations can proceed. If a gate cannot be satisfied, all operations depending on it are blocked. Three examples illustrate the concept:
   - *Identifier resolution gate*: The task says "update status for user Alice", but the update operation requires a numeric user ID, not a name. A gate captures this requirement: the user ID must first be obtained by looking up "Alice" in the user table.
   - *State precondition gate*: The task says "assign patient to bed 301", but the assignment operation requires that bed 301 is currently unoccupied. A gate captures this requirement: the bed's occupancy status must be checked before assignment can proceed.
   - *Authorization gate*: The task says "delete the billing record", but the delete operation requires admin privileges. A gate captures this requirement: the agent must have the required permission, which may not be available in the current environment.

   Each gate also records its **provenance** (where the required information comes from):
   - `task_literal`: the task instruction directly supplies the value (e.g., "update user ID 42")
   - `unique_lookup`: a tool call deterministically finds it (e.g., username "alice" matches exactly one user)
   - `broad_lookup`: multiple records match; the agent must choose (e.g., name "John Smith" matches 3 users)
   - `prior_tool_output`: an earlier operation produces it (e.g., `create_user` returns the new user ID)
   - `agent_generated`: the agent must create the value (e.g., generate a password)
   - `unavailable`: no available tool can obtain it
3. **Dependencies**: Directed edges stating which operations must finish before others can start. For example, if operation B needs the output of operation A as input, then B depends on A. This is recorded as an edge from A to B in the operation graph.
4. **Reward groups**: A partition of the verifier checks (the success conditions for the task). Verifier checks that are controlled by the same operation or gate belong to the same group, because they will succeed or fail together. Three examples:
   - Suppose the verifier has three checks: "user status is active", "user last_login timestamp is updated", and "user notification_sent flag is true". If all three fields are written by the same `update_user` operation, the three checks belong to one group with relation type `shared_operation`.

- Suppose two verifier checks both examine properties of a user record, and obtaining that user record requires satisfying an identifier-resolution gate. If the gate fails, both checks fail. They belong to one group with relation type `shared_gate`.
- Suppose one verifier check examines a field that already has the correct value in the initial state, and no operation in the task will modify it. This check is already satisfied before the agent does anything. It belongs to a group with relation type `initially_satisfied`.

5. **Conflicts**: Pairs of operations that have ordering constraints or incompatibilities beyond simple data dependencies. Three conflict types:
   - `order`: Operation A must execute before operation B, even if B does not directly consume A's output. For example, "verify eligibility must precede commit transaction" ensures that invalid transactions are never committed, even though the commit operation does not use the verification result as a data input.
   - `mutual_exclusion`: Operations A and B cannot both execute in the same task. For example, "approve request" and "reject request" are mutually exclusive: completing the task requires executing one or the other, but not both.
   - `destructive`: Operation A removes the preconditions needed by operation B. For example, `delete_user` makes any subsequent `update_user` impossible because the user record no longer exists.

After extraction, we apply a deterministic abstraction procedure to the output:

1. Replace each tool name with its operation type (e.g., `get_user_by_name` → `lookup`, `update_user_status` → `update`)
2. Remove all entity identifiers (user IDs, record keys, table names)
3. Remove all database field names (e.g., `status`, `assignment_id`)
4. Replace literal values with type placeholders (e.g., "active" → `<string>`, 42 → `<int>`)
5. Retain the dependency structure, gate types, and group relations

This produces an abstract structure description. For example, a task "update user Alice's status to active" with concrete extraction `[lookup(get_user_by_name, input=Alice), update(set_user_status, target=user_id, value=active)]` becomes abstract structure `[lookup(type=entity_by_name), update(type=entity_field)]` after applying these replacements. The abstract string is TF–IDF encoded to produce g_i in Eq. 1, which computes cross-environment similarity S_{ij}^{abs} for graph edges $\mathcal{E}^{\text{cross}}$.

E.2 System Prompt

The prompt constrains the model to use fixed vocabularies for operation types, gate types, and group relations. JSON structure is enforced by `response_format={"type":"json_object"}` in the API call.

System Prompt

```
You extract factual static topology from tool-agent tasks.  You
receive no rollout, trajectory, action trace, model output, reward
observation, success rate, or variance.  Do not predict any of them
and do not assign difficulty, probability, capability, or high/low
labels.

Build two connected structures
  from the supplied task, initial state, exact verifier checks,
  tool catalog, and selected real tool implementations:
  1.  execution topology:  the minimum correct operations, their
     dependencies, and factual prerequisite gates such as
     identifier provenance, state preconditions, ordering,
     authorization, exact arguments, and missing tool affordances;
  2.  reward topology:  a partition of verifier check IDs into
     groups that are controlled by the same operation or
     prerequisite gate.  Check weights are not yours to estimate;
     downstream code computes them from the exact check IDs.

A gate is a prerequisite
whose resolution can jointly enable or block one or more
operations/checks.  Do not create a gate for every ordinary literal.
Distinguish a direct task literal, unique lookup, broad lookup,
prior tool output, agent-generated value, and genuinely unavailable
value.  Mark reachability unsupported only when the supplied
tools/implementation establish that no path exists; otherwise use
unknown.  Describe topology, not likely model behavior.

The initial_state_summary
  is evidence available to this static analyzer, not information
  shown to the agent.  Never mark an identifier direct merely
  because it appears in that summary.  From the agent's execution
  perspective, an identifier is task_literal only when the task
  supplies the execution-ready value; otherwise classify the
  actual tool path as unique_lookup, broad_lookup,
  prior_tool_output, inferred, or unavailable.  Use the initial
  state only to verify that records, matches, and preconditions
  exist and to determine whether a supplied lookup can resolve
  them.
```

System Prompt

Use these enums exactly:

- operation.family: ['aggregate', 'authorize', 'create', 'delete', 'link', 'lookup', 'other', 'report', 'schedule', 'transition', 'update', 'validate']
- gate.type: ['actor_authorization', 'argument_exactness', 'capacity_or_eligibility', 'cross_entity_join', 'generated_identifier', 'identifier_resolution', 'ordering', 'other', 'state_precondition', 'temporal_conversion', 'termination_horizon', 'tool_affordance']
- gate.provenance: ['agent_generated', 'broad_lookup', 'inferred', 'none', 'prior_tool_output', 'task_literal', 'unavailable', 'unique_lookup', 'unknown', 'wall_clock']
- gate.reachability: ['ambiguous', 'direct', 'indirect', 'unknown', 'unsupported']
- gate.observability: ['explicit_result', 'silent', 'state_query', 'unknown']
- gate.recoverability: ['irreversible', 'retryable', 'unknown']
- reward_group.relation: ['fixed_unreachable', 'independent', 'initially_satisfied', 'mixed', 'shared_gate', 'shared_operation']
- conflict.kind: ['destructive', 'mutual_exclusion', 'order', 'preservation']

Return JSON with key "tasks".

Every task item must have exactly this structure:

```
{
  "task_id": "...",
  "operations": [
    {"id":"o0","family":"lookup","description":"...","tool_candidates":["..."],
      "depends_on_operations":[],"gate_ids":["g0"],"affected_checks":["c0"]}
  ],
  "gates": [
    {"id":"g0","type":"identifier_resolution","description":"...",
      "provenance":"unique_lookup","reachability":"indirect",
      "observability":"explicit_result","recoverability":"retryable",
      "depends_on_gates":[],"affected_operations":["o0"],"affected_checks":["c0"
  ]}
  ],
  "reward_groups": [
    {"id":"r0","check_ids":["c0"],"shared_gate_ids":["g0"],
      "producer_operations":["o0"],"relation":"shared_gate"}
  ],
  "conflicts": [{"left_operation":"o0","right_operation":"o1","kind":"order"}],
  "independent_branches": [
    {"id":"b0","operation_ids":["o0"],"check_ids":["c0"]}
  ],
  "uncertainties": ["short factual ambiguity in the supplied static evidence"]
}
```

System Prompt

```
Rules:
- Use only supplied check IDs and tool names.
- reward_groups must partition every verifier check exactly once.
  Group checks only when they share a concrete gate/producer;
  otherwise use singleton or independent groups.
- A check already true in the untouched initial state belongs in an
  initially_satisfied group unless later required operations can
  invalidate it.
- Operations describe the shortest correct state-changing/query
  plan, not speculative retries.
- IDs must be local sequential identifiers o0..., g0..., r0...,
  b0....
- Keep descriptions concise and case-specific.  At most 30
  operations, 30 gates, 30 reward groups, 20 conflicts, 20 branches,
  and 5 uncertainties per task.
- Return JSON only.
```

E.3 User Message Format

The user message is structured JSON with no additional natural language. Tasks from the same environment share a `shared_environment` object:

User Message Format

```
{
  "shared_environment": {
    "environment": "<environment class name>",
    "summary": "<prose summary>",
    "state_space": "<state definition>",
    "constraints": "<rules>",
    "tool_catalog": [
      {"name": "...", "description": "...", "required": [...],
       "arguments": [...], "signature": "..."}
    ]
  },
  "tasks": [
    {
      "task_id": "<identifier>",
      "task": "<instruction>",
      "initial_state_summary": {<generated by summarize_initial_state>},
      "verifier_checks": [
        {"check_id": "c0", "requirement": "...", "initially_true": <bool>,
         "verifier_code": "..."}
      ],
      "relevant_tool_implementations": [
        {"name": "...", "signature": "...", "documentation": "...",
         "implementation": "..."}
      ]
    }
  ]
}
```

F QUALITATIVE CASE STUDIES

This appendix presents three cases from the Qwen3-4B run, one per subsection. Each case is written as the scheduler saw it: the task, the evidence available before the decision, the prediction that evidence produced, and the eight rollouts that followed. All three are tasks that had never been selected before, so nothing was known about the task itself and every prediction came from other environment–task pairs. Neighbor tables list the representative neighbors each relation retrieved, their similarity, the number of prior rollouts, and the variance each contributes. The deployed estimates use the full neighbor sets ($K_{\text{within}} = 6$, $K_{\text{cross}} = 12$).

F.1 WITHIN-ENVIRONMENT EDGES: ENV_141–TASK_27 AT STEP 28

Two tasks in the same environment share tools, database fields, and verifier conditions. When one of them has been rolled out, its outcome is therefore evidence about the other: a mistake in the shared edit sequence costs both of them, and the checks that depend on the same steps fail in both.

The task. InsurancePolicyManagementSystem. Acting as Carolyn Ho, amend two policies: set the active period of policy OXL-44032-F, raise its hospitalization limit to 30,000 and lower its deductible to 400, add an Outpatient item and a Dental item while removing the Dental exclusion and keeping the Elective exclusion; then transfer policy ENL-74522-K to another account, set its active period, and add a Psychiatric exclusion. Sixteen verifier checks. The task has the characteristic policy-editing shape: an authorization check on the acting account, lookups that resolve policy numbers and coverage-item identifiers, paired deletions and creations of coverage items, and field updates whose order matters because the tool rejects a duplicate coverage type.

What the scheduler knew. The task itself had never been selected. Its environment was the most heavily sampled in the pool: 56 prior groups spanning 40 distinct tasks, with mean reward variance 0.055 against a pool-wide reference of 0.026 at this step. All six within-environment neighbors had been rolled out.

Within-environment neighbor	Similarity	Observations	Last step	Variance	Weight
task_5	0.554	2	26	0.0146	0.125
task_38	0.544	1	19	0.0077	0.086
task_18	0.525	1	25	0.0414	0.107
task_19	0.508	1	11	0.0000	0.056
task_17	0.506	2	21	0.1863	0.118
task_46	0.506	1	27	0.1084	0.112

The neighbors are not merely similar-sounding requests. They call the same functions on the same tables as the target, and all of them are subject to the same ordering constraint that the coverage-item tool rejects a duplicate coverage type.

Neighbor	Tools shared with the target	Checks
task_17 (var. 0.186)	add_coverage_item, remove_exclusion, update_coverage_limit_or_deductible, update_policy_active_dates	12
task_46 (var. 0.108)	add_coverage_item, remove_exclusion, update_coverage_limit_or_deductible, update_policy_active_dates	8
task_5 (var. 0.015)	add_coverage_item, remove_exclusion, update_coverage_limit_or_deductible, update_policy_active_dates	9

The prediction. The within-environment relation averages these six neighbors to 0.05. The cross-environment relation, which retrieves tasks from other environments by shape, averages to 0.02, and the global reference is 0.03. The two are fused with equal weight (Eq. 4) to 0.04.

The selection. Exploitation phase, batch rank 38, predicted percentile $q = 0.9747$, predicted mean reward 0.579 and predicted dispersion 0.194.

What the rollouts showed. $(0.1875, 0.1875, 0.1875, 0.1875, 0, 0.750, 0.750, 0.625)$; mean 0.359, variance 0.078, against a batch mean realized variance of 0.036 at this step. Seven of the eight trajectories form two clear behavioral groups, with one zero-reward exception, and the difference between them is not a matter of degree but of whether the trajectory attempts the edit sequence at all. Four trajectories call only `is_authorized_user_for_policy`, one or two calls each, and then stop. They pass three of the sixteen checks: the two authorization checks, and c10, which asks whether the Elective exclusion has been left in place — a check that a trajectory passes by doing nothing. The other thirteen checks fail. The three high trajectories carry out the full sequence of lookups, coverage-item insertions, exclusion removals, and the policy transfer, reaching 12, 12, and 10 of 16 checks, with what remains failing on the second policy's start date and its transfer. The single zero trajectory makes six calls, including `update_policy_active_dates` and `update_coverage_limit_or_deductible`, and then ends before reaching the verifier.

F.2 Cross-Environment Edges: env_190-task_8 at Step 40

Two tasks in different domains can ask for the same kind of work. Reconcile a patient's allergy list, clean up a member's mood history, and approve two expense claims have nothing in common in what they are about, but all three ask the agent to identify a record from the request, establish that it is allowed to touch it, and then apply a short list of edits to that record and nothing else. When two tasks are put together that way, they also tend to be hard or easy in the same places, which is what makes one informative about the other despite the different subject matter.

The task. ElectronicHealthRecordSystem. As Dr. Marta Fong, reconcile the allergy list of a patient: confirm authorization, locate the patient by demographics, update an existing Sulfa Drugs allergy, remove a Latex entry, add Adhesive Tape and Penicillin entries with exact reaction types and confirmation statuses, and leave no duplicates. Nothing can be edited until the patient has been located and the doctor's permission has been confirmed, so a single lookup near the start decides whether the rest of the task is reachable at all.

What the scheduler knew. No own history. The environment had been sampled 48 times over 26 distinct tasks, but the four nearest same-environment tasks had observed variances of 0.0000, 0.0123, 0.0032, and 0.0000: the earlier tasks here, which asked for the same kind of working over a patient's records, had all come out the same way. The four nearest tasks outside the environment sit in a mood tracker, an emergency-response app, an expense system, and a video-conferencing scheduler. They involve different records, different tools, and different verifiers, and none of them is about allergies.

Neighbor	What it asks the agent to do	Variance
`env_177-task_21`	delete one mood entry, edit a second, add a third	0.1160
`env_164-task_11`	cancel one session, reschedule another, revise its participants	0.0830
`env_175-task_35`	report, resolve, retract, and update incidents	0.0381
`env_161-task_32`	assign an approver, verify a document, advance a claim	0.0256

What these tasks share, despite the different subject matter, is the work they ask for: find the record the request names, establish that the agent is allowed to touch it, then change that record in a few specific ways. The target reads the acting user's rights, locates one patient, and edits that patient's allergy entries, changing one, removing another, and adding two. `env_177-task_21` does the same thing to a member's mood history: one entry deleted, one edited, one added. `env_164-task_11` works on sessions in the same manner, cancelling one and rescheduling another while revising its participant list. `env_161-task_32` advances a claim by assigning an approver, verifying a document, and moving its status forward, and `env_175-task_35` reports, resolves, retracts, and updates incidents.

The consequence of this arrangement shows up in the target's own rollouts. Because the four allergy edits all belong to the same patient and every tool takes that patient identifier, a rollout that cannot resolve the patient cannot make any of the four: five of the eight never do and score zero, while the three that succeed are separated only by which of the edits they finish. The neighbor tasks in the table are put together the same way, and the variance beside each is the variance those tasks actually produced.

This is why these tasks can inform one another even though none of them is about allergies. What is compared is not what the records mean or what the tools are called, but the shape of the work: which record has to be found before anything can happen, what is done to it, and how the task is arranged around it.

The prediction. The cross-environment relation averages these neighbors to 0.04. The within-environment relation averages to 0.02 from neighbors whose observed variances are 0.0000, 0.0123, 0.0032, and 0.0000. Those near-zero variances are not missing information: they report that the earlier ElectronicHealthRecordSystem tasks were solved the same way every time. What they cannot report is whether a different patient's allergy list, with different entries and different conflicts, will be as easy, because every task they summarize is drawn from this one environment. The cross-environment neighbors cover the tasks in other environments that are put together the same way, and their variances — 0.0830, 0.0381, 0.0256, 0.1160 — say how much such a task tends to separate the policy. The two are fused with equal weight to 0.03 (Eq. 4).

The selection. Exploitation phase, batch rank 33, $q = 0.9786$, predicted mean reward 0.562 and predicted dispersion 0.136.

What the rollouts showed. $(0.2143, 0.2143, 0.7857, 0, 0, 0, 0, 0)$; mean 0.152, variance 0.066, against a batch mean realized variance of 0.038 at this step. All five zero trajectories end before the verifier after hitting the 40-turn limit, and each one dies in the same place: alternating `get_patient_by_demographics` and `verify_user_authorization` across 40, 40, 40, 40, and 33 calls without ever obtaining a usable patient identifier or authorization result, so no downstream edit is attempted. The single 0.7857 trajectory clears the identity and authorization steps, calls `get_all_patients` and `list_patient_allergies`, and completes the allergy edits for 11 of 14 checks. The two 0.2143 trajectories reach the verifier having satisfied only the three identity and authorization checks. The failure mode is the one the cross-environment neighbors pointed to: in all five, a single unresolved lookup blocks every downstream check at once.

F.3 Both Relations Contributing: env_156-task_5 at Step 4

The two relations are used together rather than as alternatives. On a task where both find observed neighbors, the fused estimate is what determines where the task lands in the ranking.

The task. BloodDonationAppointmentSystem, early in training. Reschedule two donors' appointments at their existing clinics, release the vacated slots and block the new ones, release a slot from a cancelled appointment, confirm a fourth donor's eligibility, and create his appointment. Twelve checks. The shape is the many-lookup one typical of early cold-start batches: the agent must resolve facility names and time slots before it can change any state, so a small number of edits sits behind a long chain of lookups.

What the scheduler knew. No own history, and the environment itself had only five prior groups over four tasks, so its own tasks offer little. Both relations nonetheless retrieve recorded neighbors.

Neighbor	Relation	Similarity	Observations	Variance	Weight
`env_156-task_39`	within	0.450	1	0.1073	0.131
`env_156-task_10`	within	0.372	2	0.1093	0.184
`env_143-task_47`	cross	0.630	1	0.2013	0.065
`env_145-task_34`	cross	0.614	1	0.1552	0.055
`env_157-task_8`	cross	0.619	1	0.1036	0.063

The two within-environment neighbors share this environment's clinic and time-slot tools. The cross-environment neighbors come from hospital patient management, a review platform, and a licensing board, and share the many-lookup pattern in which a broad lookup by name decides whether anything downstream can proceed.

The prediction. The within-environment relation gives 0.07 and the cross-environment relation 0.10, fusing to 0.09 against a global reference of 0.06 (Eq. 4).

The selection. Exploitation phase, batch rank 40, $q = 0.9861$, predicted mean reward 0.505 and predicted dispersion 0.387.

What the rollouts showed. $(1, 1, 0.9167, 0.9167, 0.0833, 0.0833, 0.0833, 0)$; mean 0.510, variance 0.202, twice the batch mean realized variance of 0.091 at this step. The two perfect trajectories execute the full schedule in 24 and 26 calls and pass all twelve checks. The two 0.9167 trajectories differ from them only on c9, the creation of the fourth donor's appointment. The three 0.0833 trajectories carry out the rescheduling blocks and then fail the downstream checks. The zero trajectory spends its 40 turns alternating `list_locations` and `get_location_by_name` without ever resolving a facility name — the same lookup bottleneck the cross-environment neighbors point to.