CAPER: CLOSING THE LOOP BETWEEN LLM ACTION PRIORS AND REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) often requires extensive environment interaction, while large language models (LLMs) can provide action priors that guide exploration using pretrained knowledge. However, fixed LLM priors may be mismatched with the environment and cannot adapt when grounded failures reveal unsuitable guidance. We propose Closed-loop Action Prior Enhancement for Reinforcement Learning (CAPER), which alternates between RL policy learning and failure-guided action-prior refinement. Low-return trajectories identify weaknesses in the current policy–prior pair, which guide the retrieval of relevant task knowledge and the generation of revised prompt candidates. Candidate prompts are selected either through direct return evaluation or through agreement with actions in high-return trajectories, trading evaluation fidelity for environment-interaction cost. Across five tasks from three environments and multiple RL settings, CAPER consistently improves over baselines under total environment-interaction budgets. The implementation is available at `https://anonymous.4open.science/r/CAPER-FD27`.

1 INTRODUCTION

Reinforcement learning (RL) improves decision policies through trial-and-error interaction with the environment (Sutton & Barto, 2018). However, acquiring sufficient experience can require substantial environment interaction, particularly in complex or sparse-reward tasks (Dulac-Arnold et al., 2020). Recently, large language models (LLMs) have emerged as a complementary source of prior knowledge, providing semantic and procedural guidance before the RL agent has accumulated sufficient task-specific experience (Brown et al., 2020; Zhao et al., 2026). In this context, several methods use frozen LLMs as action priors to propose a restricted set of candidate actions for RL exploration. For example, DQN-Prior (Yan et al., 2025b) uses a frozen LLM with a fixed prompt to generate candidate actions from which the RL policy selects. Such priors can focus exploration on promising actions and reduce unnecessary interaction when they are well aligned with the environment. However, when the prior is mismatched, it may favor unsuitable actions or even exclude actions that are essential for task progress.

However, existing LLM-prior-based RL methods often exhibit an *asymmetric* information flow between the RL policy and the LLM prior. Environment interaction continually updates the RL policy, whereas the prompt-conditioned prior can remain fixed even when repeated failures reveal that its guidance is unsuitable. Consequently, grounded experience improves how the policy selects among proposed actions, but does not necessarily improve the prior that determines which actions are proposed in the first place. Recent methods partially address this limitation by using interaction memory to refine LLM priors or rollout feedback to adapt prompts for policy exploration (Yan et al., 2025a; Jiang et al., 2026). Nevertheless, they do not explicitly connect the evolving failures of an RL policy with external task knowledge to refine the action prior used in subsequent policy learning.

Human problem solving suggests a natural way to close this loop. When people encounter failures in an unfamiliar task, they rarely discard all of their prior knowledge or simply repeat the same strategy. Instead, they first use the failure to identify what went wrong. They then consult relevant instructions or draw on prior experience to revise their strategy before trying again. Importantly, a failure may reveal *what went wrong* without directly revealing *what should be done instead*. In such cases, external knowledge can provide the missing task-specific information needed to correct the

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

underlying strategy. This process suggests that grounded RL failures can serve not only as learning signals for the policy, but also as cues for retrieving relevant knowledge and refining the LLM action prior that guides future interaction.

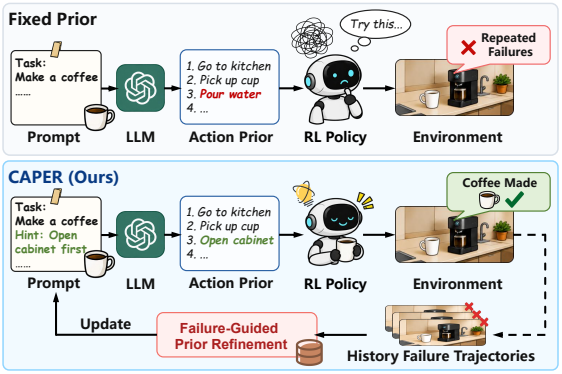


Figure 1: Illustration of fixed LLM action priors and our proposed CAPER.

Inspired by this process, we propose **C**losed-loop **A**ction **P**rior **E**nhancement for **R**einforcement Learning (CAPER), a framework that couples RL policy learning with an adaptive LLM action prior for interaction-efficient RL. As illustrated in Figure 1, CAPER keeps the LLM parameters fixed and alternates between policy learning and prior refinement. In each refinement round, the RL policy first learns for a period under the current prompt-conditioned action prior. Recent low-return trajectories are then used to expose weaknesses in the current policy–prior pair and provide grounded evidence for prior refinement. Our **Failure-Guided Prior Refinement** mechanism analyzes these failures, retrieves relevant external task knowledge, and distills the retrieved evidence into candidate prompt revisions. After a revised prompt is selected, the resulting action prior guides the next stage of RL training, which in turn produces new experience for subsequent refinement rounds. Through this alternating process, environment experience updates not only the RL policy but also the action prior that shapes subsequent exploration.

Selecting an effective prompt from the generated candidates is another key step in each refinement round. At each refinement round, the Failure-Guided Prior Refinement module produces multiple candidate prompt revisions, which CAPER selects using one of two complementary strategies. **(I) Return-Based Selection (RBS)** directly evaluates each candidate with the current RL policy and selects the one with the highest environment return. In contrast, **(II) Trajectory-Agreement Selection (TAS)** ranks candidates by how well their induced action priors agree with actions observed in high-return trajectories, avoiding candidate-specific environment rollouts. *CAPER-RBS* therefore provides direct return-based evaluation at the cost of additional environment interaction, whereas *CAPER-TAS* reduces this cost by using trajectory agreement as a surrogate for candidate quality. Together, the two variants provide a flexible trade-off between direct candidate evaluation and interaction efficiency while sharing the same failure-guided prior refinement process. Across five tasks and three RL settings, CAPER improves over baselines under total environment-interaction budgets, while also demonstrating zero-shot transfer across task configurations and action-prior LLMs.

2 Related Work

2.1 LLMs for Sequential Decision Making

LLMs have been increasingly used for sequential decision making through several complementary interfaces. A large body of work directly employs LLMs for reasoning, planning, and action selection (Zhang et al., 2024; Wang et al., 2024; Park et al., 2023; Yao et al., 2023; Hao et al., 2023). Other studies optimize language-model policies through reinforcement learning (Carta et al., 2023). LLMs can also support RL indirectly by providing reward signals or generating executable reward functions (Deng et al., 2025; Kwon et al., 2023; Ma et al., 2024).

More closely related to our work, action-prior methods use LLMs to propose candidate actions while leaving the final action selection to a learned value estimator or RL policy. DQN-Prior (Yan et al., 2025b), for example, combines a fixed prompt-conditioned LLM prior with a trainable RL verifier. Memory-Driven Self-Improvement further adapts the LLM prior using stored interactions and Q-value estimates, allowing updated priors to influence subsequent experience collection (Yan et al., 2025a). In contrast, CAPER uses grounded RL failures to identify weaknesses in the current action prior and retrieve external task knowledge for targeted prior refinement. This creates a feedback loop in which the action prior shapes RL experience, while the resulting experience in turn guides subsequent prior updates.

2.2 Prompt Optimization

Automatic prompt optimization improves prompts through search, textual feedback, or evolutionary procedures without updating model parameters (Zhou et al., 2023; Pryzant et al., 2023; Fernando et al., 2024). For interactive agents, Retroformer (Yao et al., 2024) learns retrospective prompts from policy feedback, while GEPA (Agrawal et al., 2026) evolves prompts using trajectories and textual feedback. Prompt-Driven Exploration further revises prompts based on rollout diagnoses (Jiang et al., 2026). These methods primarily optimize an LLM-based policy or reflective component.

In contrast, CAPER refines the LLM action prior that provides candidate actions to a separately trained RL policy. External knowledge can further support prompt refinement by providing task-specific information that may be absent from interaction feedback. Retrieval-augmented agents incorporate such external knowledge to improve decision making (Goyal et al., 2022; Zhu et al., 2024; Jain et al., 2024). In many existing methods, retrieved evidence is used directly to support individual decisions. CAPER instead uses grounded failures to identify relevant task knowledge and incorporates the retrieved evidence into prompt revisions for subsequent RL training.

3 Method

CAPER forms a closed loop between LLM action priors and RL policy learning by alternating policy optimization with prior refinement. During each stage of RL training, the current prompt-conditioned action prior proposes candidate actions that guide policy learning and environment interaction. At each refinement round, recent low-return trajectories are analyzed by the Failure-Guided Prior Refinement module, which retrieves relevant task knowledge and generates candidate prompt revisions. The candidates are then evaluated using either Return-Based Selection (RBS) or Trajectory-Agreement Selection (TAS), and the selected prompt defines the action prior for the next stage of RL training. This process repeats throughout training, allowing newly collected experience to continually inform subsequent prior refinement. Figure 2 summarizes this closed-loop process.

3.1 Problem Setup and Overview

We consider an RL agent that learns a policy π_θ in an environment with *textual state descriptions*. At time t, the agent receives a textual state description s_t and selects an action with the guidance of an LLM-generated action prior. Conditioned on s_t and prompt ϕ, the LLM induces an executable action distribution $\pi_\phi(\cdot \mid s_t)$ through generation and action projection. The LLM generates an indexed collection of K candidate actions, $\{a_t^{(1)}, \ldots, a_t^{(K)}\}$, from which the RL policy π_θ selects an action. Executing the policy under the current action prior yields trajectories $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$, where r_t denotes the reward at time t and $R(\tau)$ denotes the cumulative return. The RL learner updates θ using its available training data to improve the expected cumulative return. Appendix C summarizes the notation used in this paper.

However, the effectiveness of policy learning also depends on the quality of the candidate actions induced by prompt ϕ. When a fixed prompt is mismatched with the environment, the resulting action prior may repeatedly propose unsuitable actions or omit actions needed for task progress. To address this limitation, CAPER periodically refines ϕ using grounded trajectory feedback while continuing to optimize the RL policy. Specifically, every fixed interval of RL training, denoted by I_{adapt}, defines an opportunity to refine the prompt ϕ. This alternating process allows newly collected experience to guide subsequent action-prior refinement throughout training.

At each refinement opportunity, CAPER collects recent trajectories under the current policy π_θ and prompt ϕ. Low-return trajectories are selected as failure examples and form the failure set $\mathcal{D}^-$. These trajectories provide grounded evidence for prior refinement and are kept separate from the data used for policy updates. The Failure-Guided Prior Refinement module analyzes $\mathcal{D}^-$ to identify weaknesses in the current policy–prior pair, retrieves relevant evidence from an external task knowledge base, and generates N candidate prompt revisions. The resulting candidates are evaluated using either Return-Based Selection (RBS) or Trajectory-Agreement Selection (TAS). RBS directly evaluates candidate prompts through environment returns, whereas TAS ranks them according to their agreement with actions observed in high-return trajectories. The selected prompt becomes the new ϕ and defines the action prior for the next stage of RL training. New interaction experience then

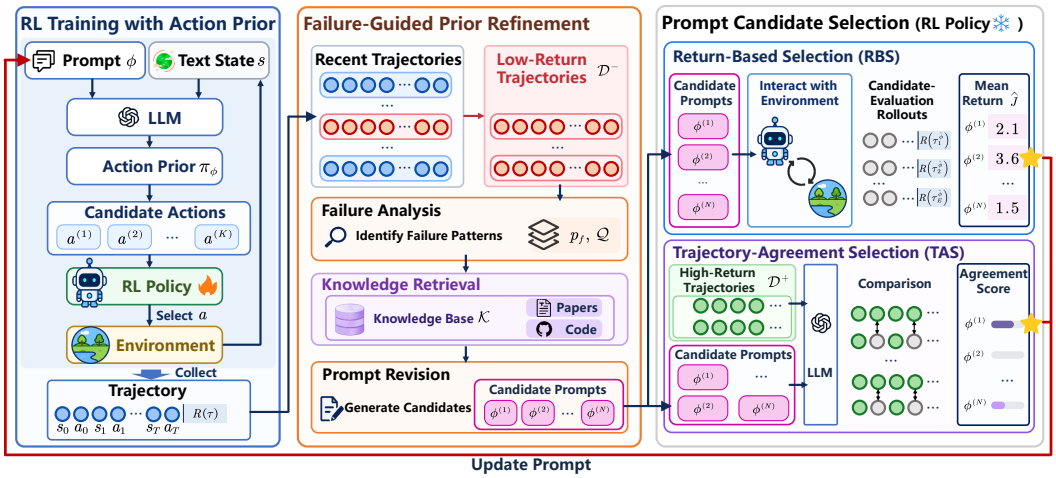


Figure 2: Overview of CAPER. Low-return trajectories guide failure-based prior refinement, while RBS or TAS selects the revised prompt to update the LLM action prior for subsequent RL training.

provides failure evidence for subsequent refinement rounds, forming a closed loop between RL policy learning and action-prior refinement.

3.2 Failure-Guided Prior Refinement

The Failure-Guided Prior Refinement module converts recent policy failures into candidate prompt revisions for the LLM action prior. It combines two complementary sources of information: **(I)** grounded failures observed under the current policy–prior pair and **(II)** external task knowledge that can provide corrective information. At each refinement round, the module considers the most recent W current-prompt trajectories and selects the F trajectories with the lowest returns. These trajectories form the failure set $\mathcal{D}^- = \{\tau_j\}_{j=1}^{F}$ and serve as the grounded evidence for subsequent refinement.

The refinement proceeds in three stages. First, a problem-analysis LLM examines $\mathcal{D}^-$ together with the current prompt ϕ_{old} to identify recurring failure patterns. It summarizes these failures into an aggregate description p_f and produces a set of retrieval keywords $\mathcal{Q}$ that characterize the missing or misapplied task knowledge. Second, ϕ_{old}, p_f, and $\mathcal{Q}$ are used to query an external task-specific knowledge base $\mathcal{K}$ constructed from task-relevant descriptions of public research articles and environment code. We denote this retrieval-augmented generation procedure by $\text{RAG}(\cdot)$, which retrieves and aggregates evidence relevant to the identified failures. The resulting evidence is represented as $\mathcal{R} = \text{RAG}(\mathcal{K}; \phi_{\text{old}}, p_f, \mathcal{Q})$. The retrieved knowledge may provide task mechanics, action preconditions, and solution-relevant procedures that are not evident from the failed trajectories alone. Third, a prompt-design LLM combines ϕ_{old}, the failure description p_f, and the retrieved evidence $\mathcal{R}$ to generate N candidate prompt revisions $\{\phi^{(1)}, \ldots, \phi^{(N)}\}$. The revisions are constrained to address the identified failures while preserving unrelated parts of the current prompt.

The module only generates candidate revisions and does not directly replace the current prompt. The resulting candidates are subsequently evaluated by RBS or TAS, and the selected candidate defines the action prior used in the next stage of policy learning. Notably, the knowledge base $\mathcal{K}$ is constructed from readily available public sources, including task-relevant research papers and environment source code. Moreover, retrieval is performed only during refinement rounds and is not invoked during step-wise action selection. Knowledge-base construction prompts are provided in Appendix B.14.1. Templates for failure analysis, retrieval, and candidate generation are provided in Appendix B.14.2.

3.3 Prompt Candidate Selection

The Failure-Guided Prior Refinement module produces multiple candidate prompt revisions at each refinement round, and CAPER must select one to define the action prior for subsequent policy learning. We consider two complementary selection strategies for CAPER that differ in how

candidate quality is estimated. Return-Based Selection (RBS) evaluates candidates through additional environment rollouts using their observed returns, whereas Trajectory-Agreement Selection (TAS) estimates candidate quality from previously collected high-return trajectories without candidate-specific interaction. These two strategies provide a trade-off between direct evaluation of downstream performance and lower environment-interaction cost.

3.3.1 RETURN-BASED SELECTION

Return-Based Selection (RBS) evaluates candidate prompts according to their empirical environment returns under the current RL policy. At each refinement opportunity, RBS first collects a small batch of rollouts under the current policy–prior pair. The mean return of these rollouts, denoted by $\bar{R}_{\mathrm{cur}}$, measures the performance of the current prompt under the current RL policy. At the first refinement opportunity, no previous prompt-update return is available as a reference. RBS therefore performs the first refinement unconditionally to establish an initial refined prior and its reference return. Operationally, $\bar{R}_{\mathrm{prev}}$ is initialized to $-\infty$, so the condition $\bar{R}_{\mathrm{cur}} > \bar{R}_{\mathrm{prev}}$ is automatically satisfied at the first refinement opportunity. After candidate evaluation, the return of the selected prompt is stored as the new $\bar{R}_{\mathrm{prev}}$. At subsequent refinement opportunities, a new refinement round is triggered only when $\bar{R}_{\mathrm{cur}} > \bar{R}_{\mathrm{prev}}$.

Once refinement is triggered, the Failure-Guided Prior Refinement module generates N candidate prompts $\{\phi^{(1)}, \ldots, \phi^{(N)}\}$. Together with the current prompt ϕ, they form the candidate set $\Phi = \{\phi, \phi^{(1)}, \ldots, \phi^{(N)}\}$. The RL policy parameters θ are kept fixed while each newly generated candidate is evaluated through additional environment rollouts. For a candidate prompt $\tilde{\phi} \in \Phi$, its empirical return is estimated as

$$\widehat{J}(\tilde{\phi}) = \frac{1}{E} \sum_{e=1}^{E} R\Big(\tau_e^{\tilde{\phi}}\Big), \tag{1}$$

where E denotes the number of evaluation episodes. $\tau_e^{\tilde{\phi}}$ denotes the *candidate-evaluation rollout* generated by the current RL policy together with the action prior induced by $\tilde{\phi}$. Then, RBS selects

$$\phi^* = \arg\max_{\tilde{\phi} \in \Phi} \widehat{J}(\tilde{\phi}),$$

and uses ϕ^* to define the action prior for the next stage of policy learning. The corresponding empirical return is stored as the new reference $\bar{R}_{\mathrm{prev}}$.

Candidate-evaluation rollouts are used exclusively for prompt selection and are excluded from RL policy updates. By evaluating each candidate through actual environment interaction, RBS provides a direct estimate of its downstream utility under the current RL policy. This directness comes at the cost of additional candidate-specific environment interaction, making RBS more suitable when such evaluation cost is affordable.

3.3.2 TRAJECTORY-AGREEMENT SELECTION

Compared with RBS, Trajectory-Agreement Selection (TAS) avoids additional candidate-specific environment interaction by evaluating candidate prompts against previously collected high-return trajectories. The underlying premise is that a useful action prior should assign high probability to actions observed in successful experience. TAS therefore measures how well the action prior induced by each candidate prompt agrees with actions in high-return trajectories. The trajectory source depends on the policy-learning setting. Under *online RL*, TAS relies on trajectories accumulated during policy training. For *offline RL*, the trajectory evidence is drawn directly from the fixed dataset.

The available trajectory source is denoted by $\mathcal{B}$, and the H trajectories with the highest returns form $\mathcal{D}^+ = \{\tau_j\}_{j=1}^{H}$. Each trajectory $\tau_j = \big((s_{j,1}, a_{j,1}), \ldots, (s_{j,T_j}, a_{j,T_j})\big)$ has return R_j and length T_j. Similarly to RBS, the current prompt $\phi^{(0)} = \phi$ is evaluated together with the N candidate prompts generated by the Failure-Guided Prior Refinement module. For each state–action pair $(s_{j,t}, a_{j,t})$, TAS queries the LLM L times using the same state $s_{j,t}$ and candidate prompt $\phi^{(i)}$, producing L action generations. The fraction of sampled actions that match the observed action $a_{j,t}$ is used to

estimate its probability $\widehat{p}$ under the candidate action prior:

$$\widehat{p}_{\phi^{(i)}}(a_{j,t} \mid s_{j,t}) = \frac{1}{L}\sum_{\ell=1}^{L} \mathbb{I}\left\{\tilde{a}_{j,t}^{(\ell)} = a_{j,t}\right\}, \quad \tilde{a}_{j,t}^{(\ell)} \sim \pi_{\phi^{(i)}}(\cdot \mid s_{j,t}).$$

Meanwhile, TAS assigns greater importance to trajectories with higher returns. Specifically, the minimum and maximum returns among the selected trajectories are denoted by $R_{\min}$ and $R_{\max}$, respectively. Accordingly, the normalized return and trajectory weight are computed as

$$\widetilde{R}_j = \frac{R_j - R_{\min} + \epsilon}{R_{\max} - R_{\min} + \epsilon}, \qquad w_j = \frac{\widetilde{R}_j}{\sum_{k=1}^{H} \widetilde{R}_k},$$

where $\epsilon > 0$ is a small numerical stabilizer. The trajectory-agreement score of candidate prompt $\phi^{(i)}$ is then defined as

$$\mathrm{Score}(\phi^{(i)}) = \sum_{j=1}^{H} w_j \left(\frac{1}{T_j} \sum_{t=1}^{T_j} \widehat{p}_{\phi^{(i)}}(a_{j,t} \mid s_{j,t}) \right). \tag{2}$$

This score first averages action agreement within each trajectory and then aggregates the trajectory-level results according to their returns.

After obtaining the score for each candidate, TAS selects the candidate with the highest trajectory-agreement score:

$$\phi^* = \arg\max_{\phi^{(i)}} \mathrm{Score}(\phi^{(i)}). \tag{3}$$

The selected prompt defines the action prior used in the next stage of policy learning. If all available trajectories have zero return, the trajectory returns provide no meaningful signal for ranking the candidates. In this case, TAS selects uniformly from the newly generated candidate prompts. Unlike RBS, TAS requires **NO** candidate-specific environment rollouts. It therefore reduces the environment-interaction cost of prompt selection by using agreement with high-return trajectories as a surrogate for downstream candidate quality.

3.3.3 Finite-Sample Selection Analysis

Both RBS and TAS aim to select the prompt with the highest underlying candidate quality. However, the corresponding population criterion cannot be evaluated exactly because only a finite number of environment rollouts or LLM generations are available. We denote the population selection criterion of prompt ϕ by $G(\phi)$ and its finite-sample estimate by $\widehat{G}(\phi)$. For RBS, $G(\phi)$ corresponds to the expected environment return, and its empirical counterpart $\widehat{G}(\phi)$ is computed from a *finite number* of candidate-evaluation rollouts as defined in Eq.(1). For TAS, the population criterion $G(\phi)$ is the trajectory-agreement score, whereas $\widehat{G}(\phi)$ is obtained using a *finite number* of LLM generations over the selected trajectories, following Eq.(2). The two selection strategies can be unified under the following empirical maximization formulation:

$$\widehat{\phi} = \arg\max_{\phi \in \Phi} \widehat{G}(\phi). \tag{4}$$

The following proposition characterizes how the estimation error between $\widehat{G}(\phi)$ and $G(\phi)$ affects the quality of the selected prompt. In particular, $\delta \in (0,1)$ controls the confidence level, while ξ bounds the largest estimation error between the empirical criterion $\widehat{G}(\phi)$ and its population counterpart $G(\phi)$ over all prompts in Φ.

Proposition 3.1 (Finite-Sample Prompt Selection)**.** *Suppose that, with probability at least* $1-\delta$*,*

$$\max_{\phi \in \Phi} \left| \widehat{G}(\phi) - G(\phi) \right| \le \xi. \tag{5}$$

Then the prompt selected by Eq.(4) satisfies

$$G(\widehat{\phi}) \ge \max_{\phi \in \Phi} G(\phi) - 2\xi. \tag{6}$$

Since the current prompt ϕ_{cur} *is included in* Φ*, it also follows that*

$$G(\widehat{\phi}) \ge G(\phi_{\mathrm{cur}}) - 2\xi. \tag{7}$$

Proposition 3.1 shows that the quality of empirical prompt selection is directly controlled by the estimation error ξ. Specifically, the selected prompt $\widehat{\phi}$ is at most 2ξ worse than the best prompt in Φ under the population criterion G. Because the current prompt is also included in the candidate set, the selected prompt cannot fall more than 2ξ below the current prompt under the same criterion. Therefore, more accurate estimation of candidate quality leads directly to more reliable prompt selection. Appendix A.1 proves Proposition 3.1. The specific form of ξ depends on the selection strategy and is derived below for RBS and TAS.

For RBS, each candidate is evaluated using E independent environment rollouts. If episodic returns lie in an interval of width B_R, Hoeffding's inequality (Hoeffding, 1963) gives

$$\xi_{\mathrm{RBS}} = B_R \sqrt{\frac{\log(2|\Phi|/\delta)}{2E}}.$$

Therefore, increasing the candidate-evaluation budget reduces the estimation error and improves the reliability of return-based prompt selection. Appendix A.2 derives the RBS bound. For TAS, the population criterion is the trajectory-agreement score over the selected high-return trajectories. With H trajectories, trajectory weights $\{w_j\}_{j=1}^{H}$, and L LLM generations for each state–action pair, the corresponding error bound is

$$\xi_{\mathrm{TAS}} = \sqrt{\frac{V_{H,L}}{2} \log \frac{2|\Phi|}{\delta}}, \qquad V_{H,L} = \sum_{j=1}^{H} \frac{w_j^2}{T_j L}.$$

The bound decreases as more LLM generations and trajectory evidence are used, yielding a more stable estimate of the trajectory-agreement criterion. Appendix A.3 derives the TAS bound. Together, these results provide a unified finite-sample characterization of RBS and TAS, with selection becoming more reliable as their respective evaluation budgets increase.

4 Experiments

We evaluate CAPER across five tasks in online value-based RL, online policy-based RL, and fixed-dataset policy learning: Overcooked (Salad), Overcooked (Tomato), TextWorld-Commonsense (TWC), VirtualHome (Food Preparation), and VirtualHome (Entertainment). Online comparisons count all environment interactions, including prompt adaptation. We measure returns and action-prior quality, ablate failure conditioning, retrieval, repeated refinement, and prompt selection, and test zero-shot generalization across task configurations and action-prior LLMs. Environment and implementation details, including interaction accounting, are provided in Appendices B.1 and B.3.

4.1 Online RL Evaluation

4.1.1 Value-Based RL

To assess failure-guided refinement in online value-based RL, we compare CAPER-DQN with DQN, DQN-Prior, and LLM-Prior under total environment-interaction budgets, including prompt-refinement interactions (Figure 3). Baseline definitions are provided in Appendix B.2. CAPER-DQN (RBS) and CAPER-DQN (TAS) reach higher final returns than both learned baselines on all five tasks. The gap is small on *Overcooked (Tomato)*, where DQN-Prior already approaches the performance ceiling with a fixed prior. The gains are larger on *Overcooked (Salad)*, TWC, and both VirtualHome tasks. These tasks place greater demands on the proposed actions because of multi-step dependencies, larger semantic action spaces, or sparse rewards.

CAPER-DQN (RBS) tends to improve earlier, whereas CAPER-DQN (TAS) catches up and reaches the highest late-stage returns on TWC. This timing is consistent with how the selectors use feedback: RBS directly evaluates candidate returns, while TAS needs informative high-return trajectories to rank prompts. Both improve on the fixed-prior baseline even when the interaction budget includes prompt refinement.

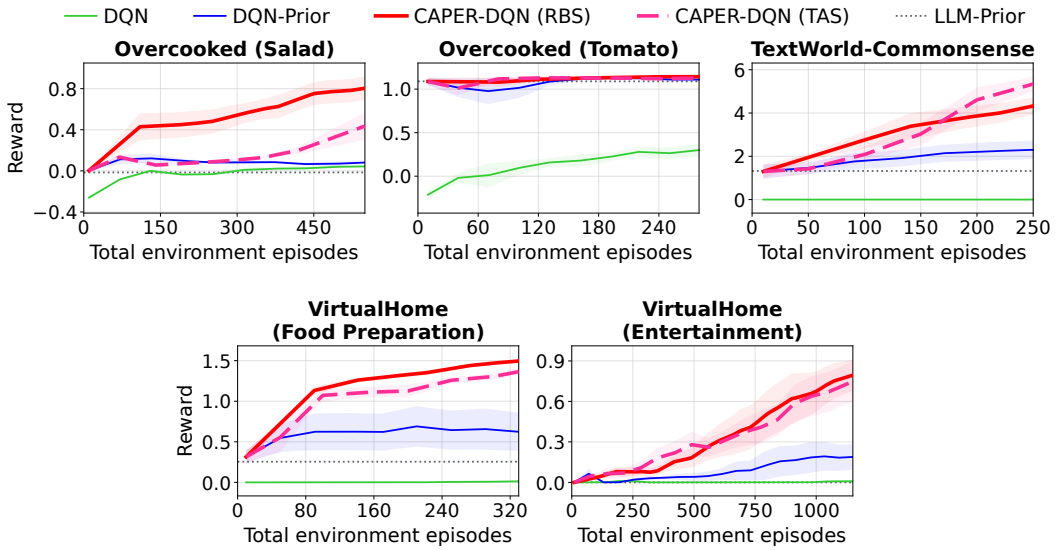


Figure 3: Value-based RL performance across five tasks under total environment-interaction budgets. Shading denotes standard error over five independent runs.

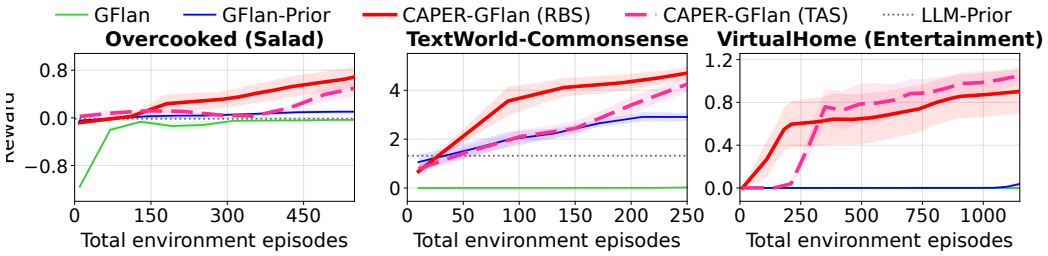


Figure 4: Policy-based RL performance across three tasks under total environment-interaction budgets.

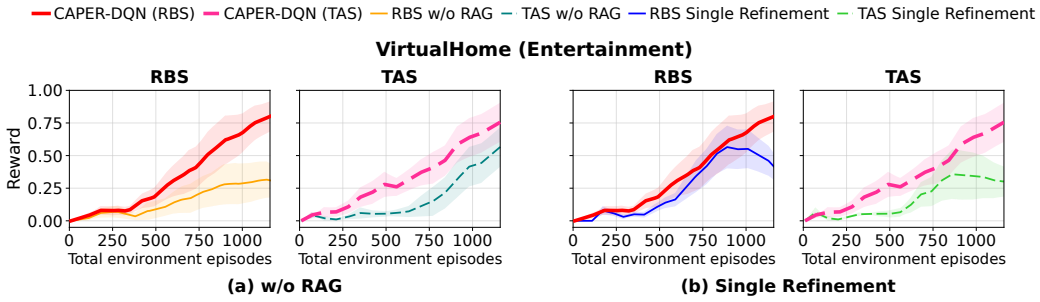


Figure 5: CAPER-DQN ablations on *VirtualHome (Entertainment)* under RBS and TAS: (a) w/o RAG and (b) Single Refinement.

4.1.2 Policy-Based RL

To test failure-guided refinement in policy-based RL, we compare CAPER-GFlan with GFlan, GFlan-Prior, and LLM-Prior using the same interaction accounting (Figure 4). Both CAPER variants outperform the learned baselines on all three tasks. On *Overcooked (Salad)*, they achieve positive returns while GFlan and GFlan-Prior remain near zero. On TWC, CAPER-GFlan (RBS) improves earlier and retains a modest final advantage over CAPER-GFlan (TAS).

The clearest separation appears on the sparse-reward *VirtualHome (Entertainment)* task. GFlan and GFlan-Prior make little progress, whereas both CAPER variants learn successful policies. CAPER-GFlan (TAS) achieves the highest late-stage return. As in the value-based experiments, adapting the

Table 1: Fixed-dataset returns across three tasks, reported as mean ± standard deviation over five runs. Arrows show gains over CQL-Prior.

Task	BC	CQL	CQL-Prior	CAPER-CQL	
				RBS	TAS
Overcooked (Salad)	-0.03 ± 0.00	0.07 ± 0.08	0.34 ± 0.10	$\mathbf{1.18 \pm 0.04}$	1.09 ± 0.18
TextWorld-Commonsense	0.00 ± 0.00	0.00 ± 0.00	2.00 ± 0.75	$\mathbf{6.00 \pm 0.00}$	5.93 ± 0.09
VirtualHome (Food Preparation)	0.00 ± 0.00	1.20 ± 0.01	0.87 ± 0.36	$\mathbf{1.49 \pm 0.02}$	1.43 ± 0.06
Average	-0.01	0.42	1.07	**2.89** $\uparrow$**1.82**	2.82 $\uparrow$1.75

action prior in response to policy failures improves returns over a fixed prior. For both value-based and policy-based RL, Appendix B.4 reports learning curves indexed by policy-training episodes.

4.2 Fixed-Dataset Policy Learning

To test CAPER with fixed-dataset policy updates, we compare CAPER-CQL with CQL, BC, and CQL-Prior. CQL updates use only the fixed dataset; CAPER collects separate interactions to adapt the prior. Both CAPER variants outperform all three baselines across the tasks (Table 1). Appendix B.5 reports the full comparison and component ablations. Training time and inference latency across all three learning settings are reported for *Overcooked (Salad)* in Appendix B.12.

4.3 Mechanism and Component Attribution

Action-Prior Quality. To test whether refinement improves the prior, we evaluate archived CAPER-DQN prompts on fixed TWC states and histories. Sampling rates increase across stages for both RBS and TAS (Figure 6). Appendix B.6 describes the inputs and per-run results. Appendix B.11 traces one prompt update with the policy fixed.

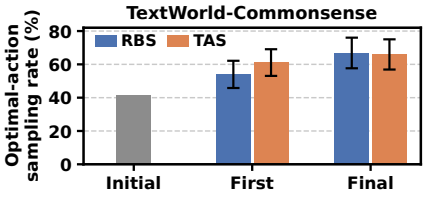


Figure 6: Optimal-action sampling rates for CAPER-DQN on TWC (mean ± standard deviation over five runs).

Failure-Guided and Repeated Refinement. To isolate retrieval and repeated refinement, we compare CAPER with w/o RAG and Single Refinement. CAPER outperforms both under RBS and TAS (Figure 5). Appendix B.5 reports the fixed-dataset ablations, Appendix B.7 the online ablations and controls, and Appendix B.13 the knowledge-base and retrieval settings.

Prompt Selection and Adaptive-Prior Baselines. To test TAS, we compare it with random prompt selection under the same budget. TAS achieves higher returns, with full results reported in Appendix B.8. CAPER also outperforms the adaptive prompt and action-prior baselines GEPA and Mem-EM w/ Tune, as detailed in Appendix B.9.

4.4 Zero-Shot Task Generalization

To test zero-shot generalization, we transfer the learned policy and optimized prompt from the six-object TWC task to an unseen seven-object task without further training or refinement. At $K = 1$, optimized prompts match or exceed the unoptimized prompt across the evaluated LLMs and both selectors. At $K = 5$, they improve three of four learner–selector settings, while $K = 10$ brings no further average gain. Appendix B.10 reports results across action-prior LLMs and candidate-set sizes.

5 Conclusion

We introduced CAPER, which alternates RL policy learning with failure-guided refinement of the LLM action prior while keeping the LLM fixed. RBS selects prompts by return, whereas TAS uses agreement with actions in stored high-return trajectories and avoids candidate-specific rollouts. Across five tasks, CAPER improves online returns under total interaction budgets and also improves fixed-dataset learning. The learned policies and prompts transfer across task configurations and action-prior LLMs without further training. Ablations support failure-conditioned retrieval and repeated refinement.

AI USE STATEMENT

We used generative AI tools to polish the writing. The authors reviewed all AI-assisted content and take responsibility for the final paper.

REPRODUCIBILITY STATEMENT

The anonymous implementation is available at https://anonymous.4open.science/r/CAPER-FD27. Sections 3.1–3.3.3 specify CAPER, including RBS and TAS, and Appendix A provides the complete proofs. Appendices B.1–B.3 describe the environments, baselines, hyperparameters, decoding settings, refinement protocol, and interaction accounting. Appendix B.5 documents the fixed datasets and full fixed-dataset results. Appendix B.13 records the knowledge-base construction and retrieval settings, while Appendices B.14.1 and B.14.2 provide the prompts used for knowledge-base construction and prior refinement. The reporting conventions for independent runs and uncertainty are stated in Appendix B.3 and in the relevant figure and table captions.

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 Carl Vondrick, Bharath Hariharan, Colin Raffel, Lerrel Pinto, Diyi Yang, and Aleksandra Faust (eds.), *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, Brazil, April 23-27, 2026*. proceedings.iclr.cc, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 3676–3713. PMLR, 2023. URL https://proceedings.mlr.press/v202/carta23a.html.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024. URL https://www.jmlr.org/papers/v25/23-0870.html.

DeepSeek-AI. DeepSeek-V3 technical report, 2024. URL https://arxiv.org/abs/2412.19437.

Yongxin Deng, Xihe Qiu, Jue Chen, and Xiaoyu Tan. Reward guidance for reinforcement learning tasks based on large language models: The LMGT framework. *Knowl. Based Syst.*, 322:113689, 2025. doi: 10.1016/J.KNOSYS.2025.113689. URL https://doi.org/10.1016/j.knosys.2025.113689.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423/.

Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. An empirical investigation of the challenges of real-world reinforcement learning. *CoRR*, abs/2003.11881, 2020. URL https://arxiv.org/abs/2003.11881.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=9ZxnPZGmPU.

Anirudh Goyal, Abram L. Friesen, Andrea Banino, Theophane Weber, Nan Rosemary Ke, Adrià Puigdomènech Badia, Arthur Guez, Mehdi Mirza, Peter Conway Humphreys, Ksenia Konyushkova, Michal Valko, Simon Osindero, Timothy P. Lillicrap, Nicolas Heess, and Charles Blundell. Retrieval-augmented reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 7740–7765. PMLR, 2022. URL https://proceedings.mlr.press/v162/goyal22a.html.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. Lightrag: Simple and fast retrieval-augmented generation. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 10746–10761. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-EMNLP.568. URL https://doi.org/10.18653/v1/2025.findings-emnlp.568.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 8154–8173. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.507. URL https://doi.org/10.18653/v1/2023.emnlp-main.507.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

Abhinav Jain, Chris Jermaine, and Vaibhav Unhelkar. RAG-Modulo: Solving sequential tasks using experience, critics, and language models. *arXiv preprint arXiv:2409.12294*, 2024. URL https://arxiv.org/abs/2409.12294.

Sunshine Jiang, John Marangola, David Zhang, Raghuram Kowdeed, Ruiyang Luo, Nitish Dashora, Richard Li, Pulkit Agrawal, and Zhang-Wei Hong. Prompt-driven exploration: Language as an exploration space for vla reinforcement learning, 2026. URL https://arxiv.org/abs/2607.08837.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=10uNUgI5Kl.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=IEduRUO55F.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing Atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Keerthiram Murugesan, Mattia Atzeni, Pavan Kapanipathi, Pushkar Shukla, Sadhana Kumaravel, Gerald Tesauro, Kartik Talamadupula, Mrinmaya Sachan, and Murray Campbell. Text-based RL agents with commonsense knowledge: New challenges, environments and baselines. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 9018–9027. AAAI Press, 2021. doi: 10.1609/AAAI.V35I10.17090. URL https://doi.org/10.1609/aaai.v35i10.17090.

Zach Nussbaum, John Xavier Morris, Andriy Mulyar, and Brandon Duderstadt. Nomic embed: Training a reproducible long context text embedder. *Trans. Mach. Learn. Res.*, 2025, 2025. URL https://openreview.net/forum?id=IPmzyQSiQE.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494/.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction, 2nd Edition*. MIT Press, 2018. URL http://www.incompleteideas.net/book/the-book-2nd.html.

Weihao Tan, Wentao Zhang, Shanqi Liu, Longtao Zheng, Xinrun Wang, and Bo An. True knowledge comes from practice: Aligning large language models with embodied environments via reinforcement learning. In *The Twelfth International Conference on Learning Representations, ICLR 2024*, 2024. URL https://openreview.net/forum?id=hILVmJ4Uvu.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Xue Yan, Zijing Ou, Mengyue Yang, Yan Song, Haifeng Zhang, Yingzhen Li, and Jun Wang. Memory-driven self-improvement for decision making with large language models. *CoRR*, abs/2509.26340, 2025a. doi: 10.48550/ARXIV.2509.26340. URL https://doi.org/10.48550/arXiv.2509.26340.

Xue Yan, Yan Song, Xidong Feng, Mengyue Yang, Haifeng Zhang, Haitham Bou-Ammar, and Jun Wang. Efficient reinforcement learning with large language model priors. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025b. URL https://openreview.net/forum?id=e2NRNQ0sZe.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract.html.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh R N, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil L Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. Retroformer: Retrospective large language agents with policy gradient optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KOZu91CzbK.

Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, Xiaojun Chang, Junge Zhang, Feng Yin, Yitao Liang, and Yaodong Yang. ProAgent: Building proactive cooperative agents with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17591–17599, 2024. doi: 10.1609/aaai.v38i16.29710. URL https://ojs.aaai.org/index.php/AAAI/article/view/29710.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Zican Dong, Yupeng Hou, Beichen Zhang, Yingqian Min, Junjie Zhang, Peiyu Liu, Xiaolei Wang, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Yiwen Hu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *Frontiers Comput. Sci.*, 20(12): 2012627, 2026. doi: 10.1007/S11704-026-60308-3. URL https://doi.org/10.1007/s11704-026-60308-3.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=92gvk82DE-.

Yichen Zhu, Zhicai Ou, Xiaofeng Mou, and Jian Tang. Retrieval-augmented embodied agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17985–17995, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Zhu_Retrieval-Augmented_Embodied_Agents_CVPR_2024_paper.html.

A PROOFS FOR FINITE-SAMPLE PROMPT SELECTION

This appendix proves Proposition 3.1 and derives the RBS and TAS estimation bounds in Section 3.3.3. We analyze one prompt-selection opportunity with the RL parameters held fixed. The population criterion is expected environment return for RBS and return-weighted trajectory agreement for TAS.

A.1 PROOF OF PROPOSITION 3.1

Proof. Let $\phi_{\mathrm{opt}} \in \arg\max_{\phi\in\Phi} G(\phi)$, and consider the event

$$\mathcal{E}_\xi = \left\{ \max_{\phi\in\Phi} \left| \widehat{G}(\phi) - G(\phi) \right| \le \xi \right\}.$$

By assumption, $\Pr(\mathcal{E}_\xi) \ge 1-\delta$. On this event, empirical maximization gives

$$\begin{aligned} G(\widehat{\phi}) &\ge \widehat{G}(\widehat{\phi}) - \xi \\ &\ge \widehat{G}(\phi_{\mathrm{opt}}) - \xi \\ &\ge G(\phi_{\mathrm{opt}}) - 2\xi = \max_{\phi\in\Phi} G(\phi) - 2\xi. \end{aligned}$$

The first and third inequalities follow from the uniform estimation bound, and the second follows from the definition of $\widehat{\phi}$. Since $\phi_{\mathrm{cur}} \in \Phi$, we also have

$$G(\widehat{\phi}) \ge G(\phi_{\mathrm{cur}}) - 2\xi.$$

Both conclusions therefore hold with probability at least $1-\delta$. □

A.2 RETURN-BASED SELECTION

Fix the RL parameters θ and the action-selection rule. Let $\mu^{(K)}_{\theta,\phi}$ denote the trajectory distribution induced by this policy and the action prior under prompt ϕ, with K candidate actions per step. The population and empirical return criteria are

$$G_{\mathrm{RBS}}(\phi) = \mathbb{E}_{\tau\sim\mu^{(K)}_{\theta,\phi}}[R(\tau)], \qquad \widehat{G}_{\mathrm{RBS}}(\phi) = \widehat{J}(\phi) = \frac{1}{E}\sum_{e=1}^{E} R(\tau_e^\phi). \tag{8}$$

For a fixed finite prompt pool Φ containing the current prompt, assume that each prompt is evaluated on E independent rollouts from $\mu^{(K)}_{\theta,\phi}$ and that returns lie in an interval of width B_R. The pool is fixed before these evaluation rollouts are collected. Independence between the evaluation batches of different prompts is not required.

Derivation of the RBS estimation bound. For any fixed $\phi \in \Phi$ and $x > 0$, Hoeffding's inequality (Hoeffding, 1963) gives

$$\Pr\Big(\left| \widehat{G}_{\mathrm{RBS}}(\phi) - G_{\mathrm{RBS}}(\phi) \right| > x \Big) \le 2\exp\left(-\frac{2Ex^2}{B_R^2} \right).$$

A union bound over Φ yields

$$\Pr\left(\max_{\phi\in\Phi} \left| \widehat{G}_{\mathrm{RBS}}(\phi) - G_{\mathrm{RBS}}(\phi) \right| > x \right) \le 2|\Phi| \exp\left(-\frac{2Ex^2}{B_R^2} \right).$$

Setting the right-hand side to δ gives

$$\xi_{\mathrm{RBS}} = B_R \sqrt{\frac{\log(2|\Phi|/\delta)}{2E}}. \tag{9}$$

Applying Proposition 3.1 with $G = G_{\mathrm{RBS}}$ and $\xi = \xi_{\mathrm{RBS}}$ proves, with probability at least $1-\delta$,

$$G_{\mathrm{RBS}}(\widehat{\phi}) \ge \max_{\phi\in\Phi} G_{\mathrm{RBS}}(\phi) - 2\xi_{\mathrm{RBS}} \ge G_{\mathrm{RBS}}(\phi_{\mathrm{cur}}) - 2\xi_{\mathrm{RBS}}. \tag{10}$$

□

Reusing Current-Prompt Rollouts. RBS uses current-prompt rollouts to decide whether to refine and to guide candidate generation, while new candidates receive separate validation rollouts. Condition on the history before a predetermined refinement opportunity, so that the current prompt, RL parameters, and action-selection rule are fixed. Assume that the current-prompt score averages E independent rollouts under this rule and that every new candidate is evaluated on E fresh independent rollouts after generation. The current-prompt and candidate-evaluation batches both contain ten episodes in our experiments.

Let $\mathcal{T}$ be the event that refinement is triggered. The current-prompt estimation error obeys the single-prompt Hoeffding bound before its rollouts are observed. For each new candidate, the same tail bound holds conditional on the current-prompt data and candidate-generation history, because its validation rollouts are fresh. Averaging these conditional bounds and taking a union bound over the current prompt and the N new candidates gives

$$\Pr\left(\mathcal{T} \text{ and } \max_{\phi\in\Phi}\left|\widehat{G}_{\mathrm{RBS}}(\phi) - G_{\mathrm{RBS}}(\phi)\right| > x\right) \le 2(N+1)\exp\left(-\frac{2Ex^2}{B_R^2}\right). \tag{11}$$

Here the candidate pool contains $N+1$ indexed prompts, including the current prompt, and probabilities are measured before the trigger decision. Taking $|\Phi| = N+1$ in Eq.(9), the joint probability of triggering refinement and violating Eq.(10) at this opportunity is at most δ, even when candidate generation depends on the current-prompt trajectories.

A.3 Trajectory-Agreement Selection

Consider the TAS branch that maximizes the trajectory-agreement score. Condition on the selected trajectories $\mathcal{D}^+ = \{\tau_j\}_{j=1}^H$, their lengths T_j, their weights $w_j \ge 0$ with $\sum_j w_j = 1$, and the finite candidate pool Φ containing the current prompt. These quantities are fixed before the LLM generations used for scoring. For each prompt ϕ, let

$$X_{j,t,\ell}^{(\phi)} = \mathbb{I}\left\{\tilde{a}_{j,t}^{(\ell)} = a_{j,t}\right\}, \qquad \tilde{a}_{j,t}^{(\ell)} \sim \pi_\phi(\cdot \mid s_{j,t}).$$

Assume that these scoring generations are independent over j, t, and ℓ for each fixed prompt. The indicators $X_{j,t,\ell}^{(\phi)}$ have conditional means $\pi_\phi(a_{j,t} \mid s_{j,t})$. The population criterion and its estimate are

$$G_{\mathrm{TAS}}(\phi) = \sum_{j=1}^{H}\frac{w_j}{T_j}\sum_{t=1}^{T_j}\pi_\phi(a_{j,t} \mid s_{j,t}), \tag{12}$$

$$\widehat{G}_{\mathrm{TAS}}(\phi) = \sum_{j=1}^{H}\frac{w_j}{T_jL}\sum_{t=1}^{T_j}\sum_{\ell=1}^{L}X_{j,t,\ell}^{(\phi)} = \mathrm{Score}(\phi). \tag{13}$$

The trajectory states are fixed under this conditioning and need not be independent.

Derivation of the TAS estimation bound. For a fixed ϕ, the error $\widehat{G}_{\mathrm{TAS}}(\phi) - G_{\mathrm{TAS}}(\phi)$ is a weighted sum of centered Bernoulli variables. With coefficients $c_{j,t,\ell} = w_j/(T_jL)$, the sum of squared ranges is

$$\sum_{j=1}^{H}\sum_{t=1}^{T_j}\sum_{\ell=1}^{L}c_{j,t,\ell}^2 = \sum_{j=1}^{H}\frac{w_j^2}{T_jL} = V_{H,L}. \tag{14}$$

Weighted Hoeffding's inequality therefore gives, for $x > 0$,

$$\Pr\left(\left|\widehat{G}_{\mathrm{TAS}}(\phi) - G_{\mathrm{TAS}}(\phi)\right| > x\right) \le 2\exp\left(-\frac{2x^2}{V_{H,L}}\right),$$

where the probability is conditional on the trajectories, weights, and candidate pool. A union bound yields

$$\Pr\left(\max_{\phi\in\Phi}\left|\widehat{G}_{\mathrm{TAS}}(\phi) - G_{\mathrm{TAS}}(\phi)\right| > x\right) \le 2|\Phi|\exp\left(-\frac{2x^2}{V_{H,L}}\right).$$

Consequently, the uniform estimation error is at most

$$\xi_{\mathrm{TAS}} = \sqrt{\frac{V_{H,L}}{2} \log \frac{2|\Phi|}{\delta}} \tag{15}$$

with conditional probability at least $1 - \delta$. Proposition 3.1 then gives

$$G_{\mathrm{TAS}}(\widehat{\phi}) \geq \max_{\phi \in \Phi} G_{\mathrm{TAS}}(\phi) - 2\xi_{\mathrm{TAS}} \geq G_{\mathrm{TAS}}(\phi_{\mathrm{cur}}) - 2\xi_{\mathrm{TAS}}. \tag{16}$$

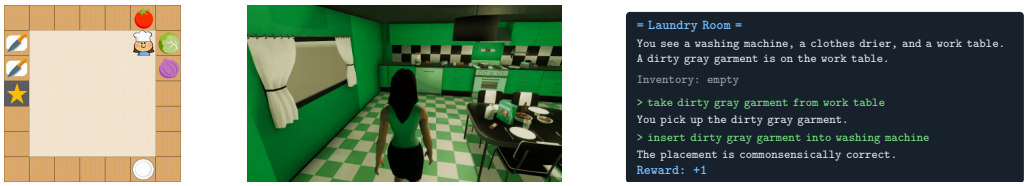
□

For fixed Φ and δ, the radius in Eq.(15) is proportional to $\sqrt{V_{H,L}}$. With equal weights $w_j = 1/H$ and equal lengths $T_j = T$, $V_{H,L} = 1/(HTL)$. For fixed trajectories and weights, the radius decreases as $L^{-1/2}$. When all available trajectories have zero return, TAS instead uses the uniform selection rule described in Section 3.3.2.

B Experimental Details and Additional Results

B.1 Simulation Environments

We evaluate CAPER in three environment families: Overcooked (Tan et al., 2024), VirtualHome (Tan et al., 2024), and TextWorld-Commonsense (Murugesan et al., 2021). Figure 7 illustrates their task interfaces. Across all three environments, the agent observes textual state descriptions rather than rendered pixels and selects semantic or macro-action candidates.



(a) Overcooked. (b) VirtualHome. (c) TextWorld-Commonsense.

Figure 7: Representative interfaces of the evaluation environments.

Panels (a) and (b) are frames from TWOSOME demonstrations (Tan et al., 2024); panel (c) shows a TextWorld-Commonsense interaction (Murugesan et al., 2021).

The three environments test different forms of decision making: grid-based manipulation with dense reward shaping in Overcooked, text-based object-location reasoning in TWC, and long-horizon household planning with sparse completion rewards in VirtualHome.

Overcooked (Figure 7a) is a partially observable, text-based kitchen environment. At each step, the observation reports the positions of visible objects, and the agent selects from up to eight macro-actions: movement in four directions, picking up, placing, chopping, and delivering. We evaluate two tasks: *Overcooked (Tomato)* requires delivery of a chopped tomato, while *Overcooked (Salad)* requires a salad containing chopped tomato and lettuce. A correct delivery gives a terminal reward of 1, with shaping rewards of $+0.2$ for a successful chop, -0.1 for an incorrect delivery, and -0.001 per step.

TextWorld-Commonsense (TWC) (Figure 7c) is a text-based house-tidying environment in which the agent places objects in commonsensically appropriate locations. Instances vary in their objects, target locations, and rooms. We train on a hard six-object task and evaluate zero-shot transfer to a separate unseen seven-object task. The agent chooses among 50 semantic text commands and receives a reward of $+1$ for each correctly placed object.

VirtualHome (Figure 7b) simulates household tasks across furnished rooms with interactable objects. The agent selects from up to 17 admissible semantic actions to complete multi-step activities. In *VirtualHome (Food Preparation)*, it must move cold pancakes from a table to the kitchen and heat

them in a microwave. In *VirtualHome (Entertainment)*, it must bring milk and chips from the kitchen to the living room, turn on the TV, sit on the sofa, and watch television. Rewards are sparse: all nonterminal rewards are zero, and completing every goal at step T yields $1 + \max(15 - T, 0)/15$.

B.2 Baselines

LLM-Prior (Yan et al., 2025b) samples actions directly from a fixed LLM action prior, without an RL policy. We also compare CAPER with online value-based and policy-based RL methods and with baselines trained on fixed datasets.

The value-based experiments compare DQN (Mnih et al., 2013) and DQN-Prior (Yan et al., 2025b) with CAPER-DQN (RBS) and CAPER-DQN (TAS). DQN-Prior keeps its LLM action prior fixed throughout training. The policy-based experiments compare GFlan (Carta et al., 2023) and GFlan-Prior (Yan et al., 2025b) with CAPER-GFlan (RBS) and CAPER-GFlan (TAS). GFlan uses Flan-T5 Small (Chung et al., 2024) as its action policy and optimizes it with Proximal Policy Optimization (PPO) (Schulman et al., 2017). We use the sub-action-space variant of GFlan-Prior, which restricts its action candidates using the same fixed LLM prior as DQN-Prior.

The fixed-dataset experiments compare CQL (Kumar et al., 2020), BC, and CQL-Prior (Yan et al., 2025b) with CAPER-CQL (RBS) and CAPER-CQL (TAS). CQL-Prior uses a fixed LLM action prior to restrict the candidates considered during value estimation and policy extraction. BC uses Flan-T5 Small as its policy network and minimizes $-\mathbb{E}_{(s,a)\sim\mathcal{D}_{\text{off}}}[\log \pi(a \mid s)]$.

The attribution study uses five additional controls. **w/o Failure Feedback** shares CAPER's task knowledge base, RL backbone, adaptation opportunities, and RBS or TAS selector, but constructs its retrieval and prompt-revision input from the task description rather than failure trajectories. **Static Full-KB Prior + DQN** supplies the complete task context to DeepSeek once before training and keeps the resulting guide fixed throughout DQN optimization.

Mem-EM w/ Tune follows Memory-Driven Self-Improvement (Yan et al., 2025a) and tunes the LLM action prior from interaction memory and Q-value estimates. **GEPA (LLM-only)** (Agrawal et al., 2026) uses the official reflective prompt optimizer with Qwen2.5-7B acting directly in the environment, without a learned DQN selector or external RAG. **GEPA + DQN** initializes DQN-Prior with the prompt from the corresponding GEPA run and keeps it fixed for 500 policy-training episodes; the DQN stage performs no prompt update, retrieval, or DeepSeek call. Both GEPA comparisons include all prompt-optimization interactions in the total budget. These comparisons test failure-conditioned retrieval against task-only retrieval, memory-based tuning, and full-knowledge initialization. They also compare prompt optimization alone with RL learning under an optimized static prompt.

Within each backbone comparison, methods use the same action model and initial prompt unless the ablation changes the available prompt information or its optimization. We use Qwen2.5-7B (Yang et al., 2024) as the primary action-prior model, chosen for its reasoning ability and inference cost. The value-based experiments encode state-action pairs with BERT-base-uncased (Devlin et al., 2019) and score them with a two-hidden-layer MLP, while the policy-based experiments optimize Flan-T5 Small. LightRAG (Guo et al., 2025) uses 768-dimensional `nomic-embed-text` embeddings (Nussbaum et al., 2025). DeepSeek-V3 (DeepSeek-AI, 2024), accessed through the `deepseek-chat` API identifier, prepares the indexed environment descriptions and handles retrieval completion, failure analysis, and candidate-prompt generation.

B.3 Implementation Details and Reproducibility

Table 2 lists the hyperparameters for CAPER and its RL backbones; shared DQN and action-selection settings follow Yan et al. (2025b) where applicable. The symbols N_{train}, I_{eval}, $T_{\max}$, η, B_{batch}, C, and U denote the training budget, evaluation interval, configured episode-step limit, learning rate, batch size, replay-buffer capacity, and DQN update interval. For Salad, the step-limit parameter is set to 30; episodes are truncated after the decision count exceeds this threshold, allowing up to 31 decisions. For DQN and PPO, N_{train} and I_{eval} are measured in environment episodes; for CQL, they are measured in optimizer steps over fixed data. Prompt adaptation uses the task-dependent interval I_{adapt}. All RL backbones share the discount γ, and DQN and CQL share the target-network coefficient τ_{target}. Task-dependent vectors follow this order: DQN uses Overcooked Salad, Overcooked

Table 2: Core experimental hyperparameters.

Method	Symbol	Value
CAPER	K	5
	N	3
	W	10
	F	3
	E	10
	H	3
	L	5
DQN	N_{train}	$(500, 200, 200, 250, 1000)$
	I_{eval}	$(50, 20, 30, 30, 50)$
	$T_{\max}$	$(30, 30, 20, 30, 30)$
	η	$(5\times10^{-4}, 5\times10^{-4}, 5\times10^{-4}, 5\times10^{-4}, 10^{-4})$
	B_{batch}	128
	C	4000
	ϵ_{DQN}	0.2
	U	5
	α_{DP}	0.1
PPO	N_{train}	$(500, 200, 1000)$
	I_{eval}	$(50, 30, 50)$
	$T_{\max}$	$(30, 20, 30)$
	η	10^{-6}
	B_{batch}	64
	N_{epoch}	4
	λ	0.99
	ϵ_{clip}	$(0.2, 0.2, 0.3)$
	c_{ent}	0.01
	c_V	0.5
CQL	N_{train}	$(500, 300, 500)$
	I_{eval}	$(50, 30, 50)$
	$T_{\max}$	$(30, 20, 30)$
	η	5×10^{-4}
	B_{batch}	128
	C	20000
	α_{CQL}	0.5
All	γ	0.99
DQN/CQL	τ_{target}	0.005

Tomato, TextWorld-Commonsense, VirtualHome Food Preparation, and VirtualHome Entertainment; PPO uses Overcooked Salad, TextWorld-Commonsense, and VirtualHome Entertainment; CQL uses Overcooked Salad, TextWorld-Commonsense, and VirtualHome Food Preparation.

Inference and Decoding. We load the local Qwen action prior in bfloat16 with automatic device mapping. For each state prompt, it returns five candidate sequences with temperature 1.0, top-$p = 1.0$, and at most 10 new tokens per candidate. Each Failure-Guided Prior Refinement request contains one DeepSeek conversation with at most 1024 output tokens. Failure analysis uses temperature 0.2, while candidate-prompt generation uses temperature 1.0. LightRAG uses naive retrieval, up to four asynchronous completion calls, an 8192-token completion context, 768-dimensional embeddings, and an 8192-token embedding limit.

Prompt Templates. Appendix B.14.2 provides before-and-after examples and the complete templates for Failure-Guided Prior Refinement and RAG.

Failure-Guided Prior Refinement. At each adaptation round, CAPER collects ten complete rollouts under the current prompt, independently of task-performance evaluation. The refinement module selects the three lowest-return trajectories and favors longer trajectories when returns tie. Failure analysis turns this evidence into a query for the task-specific LightRAG index. The prompt designer combines the retrieved evidence with the current prompt to produce three candidate prompts. The knowledge base is queried only during prompt adaptation; action selection uses the knowledge distilled into the selected prompt.

RBS Selection. RBS computes the mean return $\bar{R}_{\mathrm{cur}}$ of the current policy-prior pair over the current-prompt rollouts and starts candidate selection when $\bar{R}_{\mathrm{cur}} > \bar{R}_{\mathrm{prev}}$. This mean is the score of the current prompt. Each new candidate is evaluated with the current policy over ten independent episodes.

TAS Selection. TAS invokes Failure-Guided Prior Refinement at fixed adaptation intervals and ranks the current and newly generated prompts without candidate-specific environment rollouts. For online policy learning, it uses the three highest-return trajectories from a separate history of complete training trajectories; for policy learning from a fixed dataset, it uses the three highest-return trajectories in that dataset. At each state, TAS scores a prompt by the fraction of five generations that reproduce the demonstrated action. It averages these statewise scores over time and weights each trajectory by its normalized return. If all available trajectories have zero return, TAS chooses uniformly among the new candidates. We keep task-performance evaluation episodes separate from the data used by Failure-Guided Prior Refinement, RBS, and TAS.

Value-Based Experiments. Standard DQN and CQL score the executable actions supplied by the environment, while their prior-guided counterparts score LLM-generated candidates. Following the Overcooked DQN-Prior configuration (Yan et al., 2025b), these methods use batch size 128, update interval $U = 5$, and action-selection temperature $\alpha_{\mathrm{DP}} = 0.1$, corresponding to inverse temperature $\beta_{\mathrm{sel}} = 10$. The learning rate is 5×10^{-4} except for the VirtualHome Entertainment setting reported in Table 2. Online DQN training uses ϵ_{DQN}-greedy selection with $\epsilon_{\mathrm{DQN}} = 0.2$ over the generated candidates. Evaluation and RBS candidate validation instead sample from $\mathrm{softmax}(\beta_{\mathrm{sel}} Q_\theta)$. Online value-based models use AdamW and soft target updates.

Policy-Based Experiments. We optimize the standard clipped PPO objective over the restricted candidate set and sample actions from the resulting categorical distribution. The action prior changes the available candidates but adds no KL term to the PPO loss.

Fixed-Dataset CQL Experiments. Every value-network update uses only the offline dataset, to which we add no online transitions. Trajectories collected for task-performance evaluation or prompt adaptation contribute neither gradients nor targets to CQL. The conservative-loss coefficient is $\alpha_{\mathrm{CQL}} = 0.5$ (Table 2) and is distinct from the DQN-Prior selection temperature α_{DP}. CQL and CQL-Prior access the environment only for task-performance evaluation. Both CAPER-CQL variants also collect current-prompt rollouts for Failure-Guided Prior Refinement; RBS validates new candidates online, while TAS ranks them from the fixed dataset without candidate-specific rollouts. CAPER-CQL thus learns its policy from fixed data and adapts its action prior through separate online rollouts.

Total-Interaction View. Figures 3 and 4 in the main text count every environment episode used by each method. For each run, we compute the expanding final-third return along its cumulative-interaction coordinates. The main panels end at the largest budget covered by every method and run. Curves report the mean and standard error over five independent runs without additional reward smoothing.

Policy-Update Interaction View. Figures 8 and 9 in the appendix plot returns against RL-training episodes, counting only environment data eligible for policy updates. This view compares how efficiently methods use policy-training interactions. Fixed-dataset CQL results report the mean and standard deviation over independent runs.

Environment-Interaction Accounting. Total-interaction budgets include online policy-learning episodes, task-performance evaluation, current-prompt data collection, and RBS candidate validation. Fixed-dataset CQL updates add no environment interactions. RBS includes an initial refinement, whereas TAS begins after the first policy-learning interval once training trajectories are available. Table 3 summarizes the accounting rules; task-performance evaluation remains separate from prompt adaptation.

For Single Refinement, we count the prompt-adaptation cost once, when the optimized prompt is produced.

Table 3: Environment-interaction components included in the total budget.

Component	Environment episodes counted
Online policy learning	One per DQN/PPO training episode
Fixed-dataset CQL updates	0
Task-performance evaluation	10 per evaluation interval
Current-prompt data (RBS/TAS)	10 per adaptation round
Candidate validation (RBS)	$N \times E = 3 \times 10 = 30$ per triggered refinement
Candidate validation (TAS)	0

B.4 Policy-Update Interaction Views

To measure policy-learning sample efficiency separately from prompt-adaptation cost, we plot the main online runs against episodes eligible for policy updates. Figures 3 and 4 in the main text count all environment episodes; Figures 8 and 9 here count only policy-training episodes. CAPER retains its advantage under both accounting schemes.

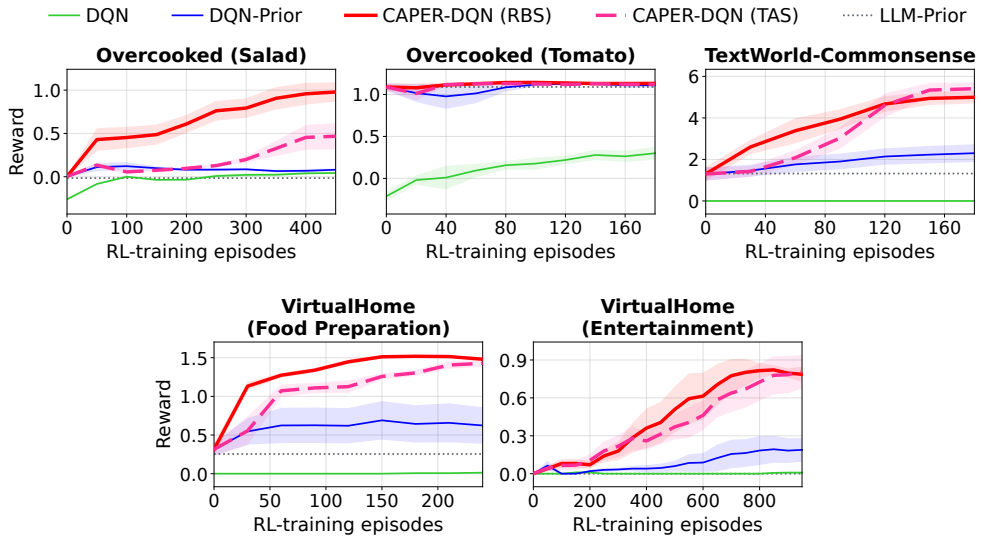


Figure 8: Value-based RL performance versus policy-training episodes.

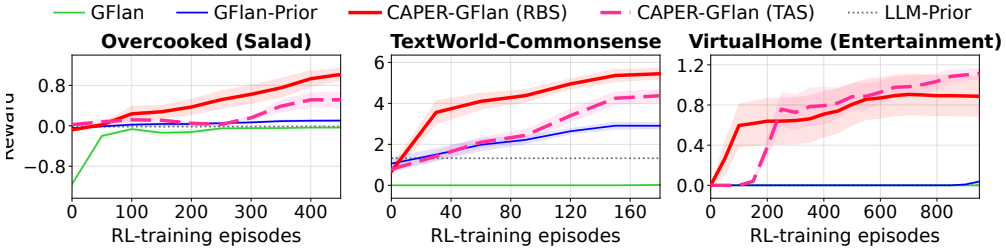


Figure 9: Policy-based RL performance versus policy-training episodes.

B.5 Full Fixed-Dataset Results

To examine which components improve fixed-dataset policy learning, we compare CAPER-CQL with its ablations using the same fixed datasets. Table 4 lists the dataset composition.

Table 4: Composition of the fixed datasets used for CQL.

Task	Total examples	Good examples	Bad examples
Overcooked (Salad)	1,030	503	527
TextWorld-Commonsense	1,020	1,020	0
VirtualHome (Food Preparation)	999	503	496

The full CAPER-CQL variants achieve the highest returns across all three tasks (Table 5). Removing RAG or limiting adaptation to a single refinement lowers returns.

Table 5: Fixed-dataset returns and component ablations (mean $\pm$ standard deviation over five runs).

Configuration				Mean episodic return ↑		
Learner	**Prompt**	**Refinement**	**Selector**	**Overcooked (Salad)**	**TextWorld-Commonsense**	**VirtualHome (Food Preparation)**
BC	None	–	–	-0.03 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
CQL	None	–	–	0.07 ± 0.08	0.00 ± 0.00	1.20 ± 0.01
	Fixed	–	–	0.34 ± 0.10	2.00 ± 0.75	0.87 ± 0.36
	Refined	w/o RAG	RBS	0.73 ± 0.15	2.43 ± 0.66	1.33 ± 0.03
			TAS	0.61 ± 0.18	3.03 ± 0.83	1.36 ± 0.03
		Single Refinement	RBS	0.93 ± 0.17	4.77 ± 1.74	1.30 ± 0.05
			TAS	0.73 ± 0.37	3.50 ± 1.42	1.35 ± 0.11
		Failure-Guided	RBS	$\mathbf{1.18 \pm 0.04}$	$\mathbf{6.00 \pm 0.00}$	$\mathbf{1.49 \pm 0.02}$
			TAS	1.09 ± 0.18	5.93 ± 0.09	1.43 ± 0.06

B.6 Action-Prior Quality Across Prompt Updates

To measure how action-prior quality changes with refinement, we evaluate prompts saved during five independent runs of CAPER-DQN (RBS) and five of CAPER-DQN (TAS) on TextWorld-Commonsense. We extract 1,000 evaluation inputs from saved TWC training trajectories using the same data-preparation procedure as for our offline RL datasets. Each input contains the same information and uses the same format as the action-prior LLM receives during training, including the state and its history. We evaluate every prompt on this fixed set.

We use task-specific rules to label an optimal-action set $\mathcal{A}_i^\star$ for the state in each input i. A state can have multiple optimal actions. Under prompt ϕ, the action-prior LLM samples $K = 5$ candidate actions $a_{i,1}(\phi), \ldots, a_{i,K}(\phi)$ for each of the $N_{\text{eval}} = 1{,}000$ inputs. A sampled action counts as optimal if it belongs to $\mathcal{A}_i^\star$. The *optimal-action sampling rate* is

$$S(\phi) = \frac{1}{N_{\text{eval}} K} \sum_{i=1}^{N_{\text{eval}}} \sum_{j=1}^{K} \mathbf{1}[a_{i,j}(\phi) \in \mathcal{A}_i^\star]\,. \tag{17}$$

We report this rate as a percentage. It measures the quality of the prior's sampled actions before DQN selects one.

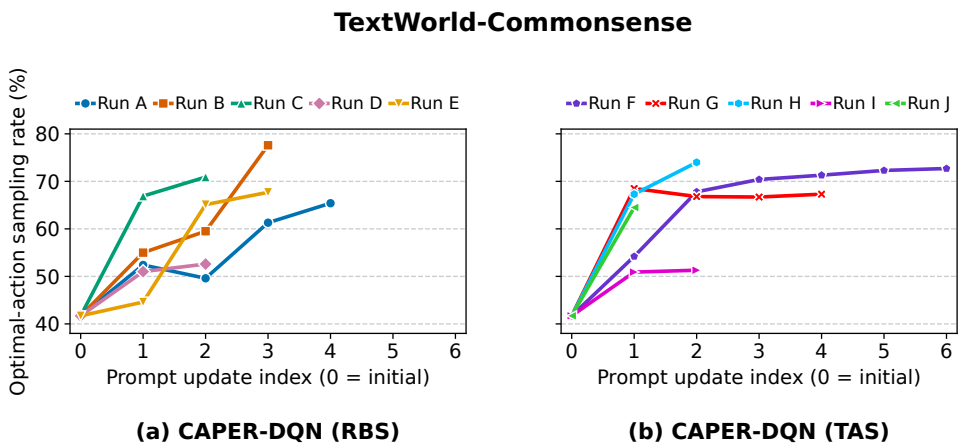


Figure 10: Optimal-action sampling rates across CAPER-DQN prompt updates on TWC.

Figure 10 shows each run's full sequence, with prompt updates on the horizontal axis. Under both RBS and TAS, optimal-action sampling rates trend upward, and every final prompt improves over the shared initial prompt. Figure 6 summarizes the initial, first-updated, and final stages.

B.7 Failure-Conditioned Retrieval and Repeated Refinement

To test the contributions of failure feedback, retrieved knowledge, and repeated refinement, we compare CAPER with matched controls. The **w/o Failure Feedback** variant builds retrieval queries and prompt revisions from the task description alone. It shares CAPER's knowledge base, RL backbone, selector, prompt generator, and update opportunities. Conditioning on current failures improves returns under both RBS and TAS (Figure 11(a)). CAPER also outperforms Static Full-KB, which distills the complete task knowledge into a fixed prompt before training (Figure 11(b)).

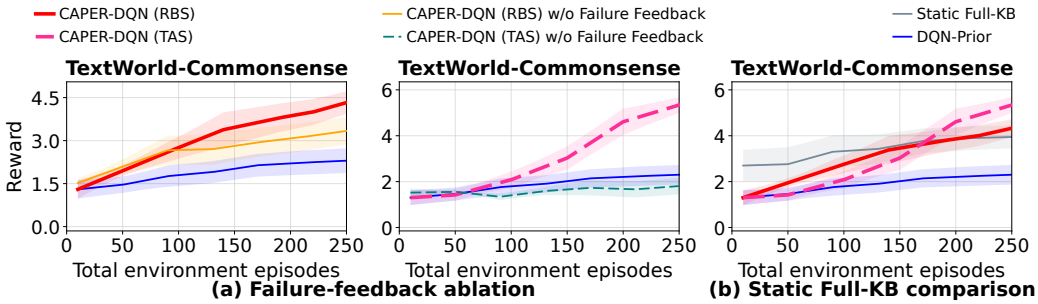


Figure 11: Failure-conditioned retrieval and repeated-refinement controls on TextWorld-Commonsense.

To test the contribution of retrieved knowledge, we remove RAG while retaining failure analysis, prompt revision, and RBS or TAS selection. The full variants outperform their w/o RAG counterparts in the value-based, policy-based, and fixed-dataset settings (Figures 12 and 13 and Table 5).

To test whether repeated refinement improves returns, we compare CAPER with Single Refinement, which refines the prompt at the first adaptation opportunity and then keeps it fixed. Repeated refinement yields higher final returns in both the online and fixed-dataset comparisons (Figures 14 and 15 and Table 5).

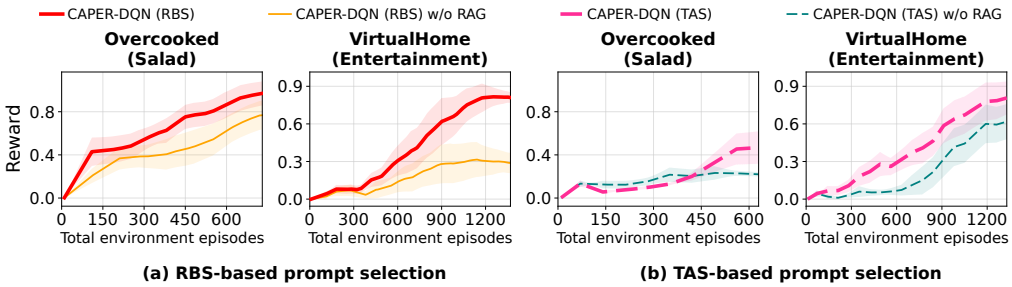


Figure 12: RAG ablations for CAPER-DQN (RBS) and CAPER-DQN (TAS).

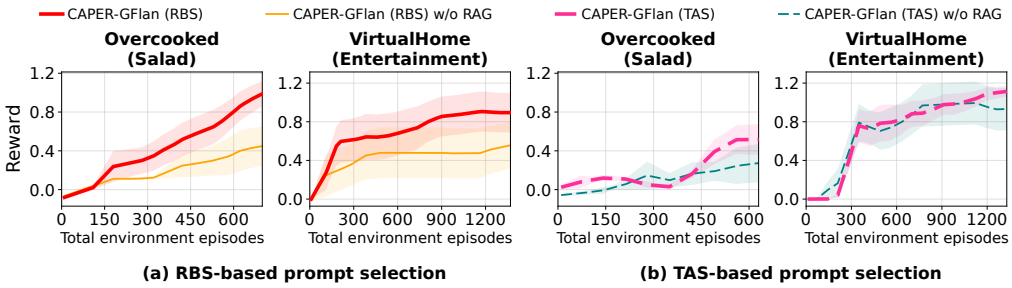


Figure 13: RAG ablations for CAPER-GFlan (RBS) and CAPER-GFlan (TAS).

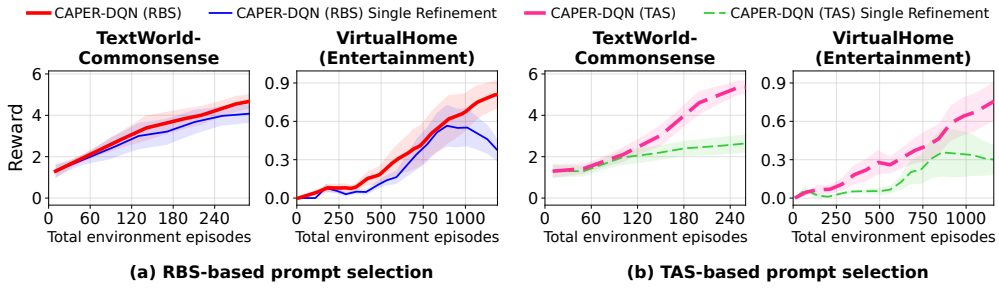


Figure 14: Single Refinement ablations for CAPER-DQN (RBS) and CAPER-DQN (TAS).

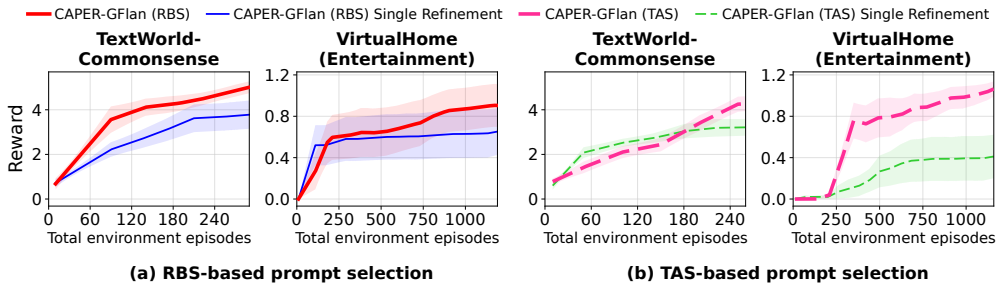


Figure 15: Single Refinement ablations for CAPER-GFlan (RBS) and CAPER-GFlan (TAS).

B.8 Trajectory-Agreement Selection

To test whether trajectory agreement helps select prompts, we replace TAS with uniform random selection. Failure-Guided Prior Refinement, the knowledge base, DQN, the adaptation schedule, and the interaction budget remain unchanged. CAPER-DQN (TAS) achieves a higher return with trajectory-agreement ranking than with random selection (Figure 16).

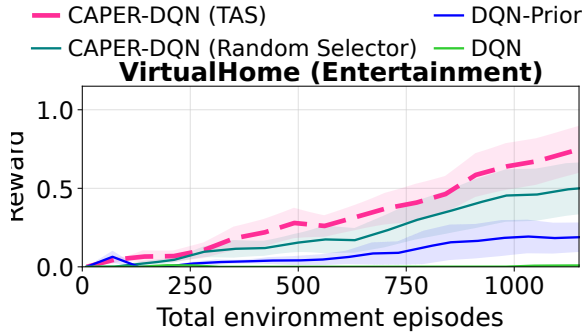


Figure 16: TAS versus uniform random prompt selection on *VirtualHome (Entertainment)*.

B.9 Adaptive Prompt and Action-Prior Baselines

To compare CAPER with other adaptive prompt and action-prior methods, we evaluate GEPA (Agrawal et al., 2026) both as a direct policy and as a frozen optimized prior for DQN, alongside Mem-EM w/ Tune (Yan et al., 2025a). CAPER-DQN (RBS), which continues to refine its prior from failures, outperforms both GEPA variants (Figure 17(a)). CAPER also achieves higher returns than Mem-EM w/ Tune on the evaluated TWC and VirtualHome tasks (Figure 17(b)).

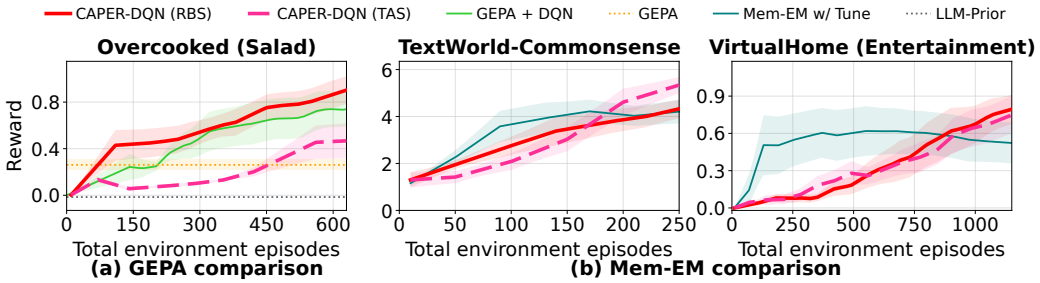


Figure 17: Comparison with adaptive prompt and action-prior baselines.

B.10 Zero-Shot Generalization

To test zero-shot task generalization, we transfer the learned policy and optimized prompt from the six-object TWC training task to an unseen seven-object task under RBS and TAS. We vary the action-prior LLM and candidate-set size while keeping the policy and prompt fixed, with no further RL training, LLM fine-tuning, or prompt refinement. Tables 6 and 7 report normalized returns for the two selectors; $K = 1_{\text{init}}$ uses the unoptimized prompt.

For both selectors, the frozen policies and optimized prompts transfer to the unseen task across the evaluated action-prior LLMs. Optimized prompts generally yield higher returns than unoptimized prompts, and $K = 5$ gives the highest returns in most configurations.

Table 6: Normalized returns for zero-shot TWC generalization under RBS. $K = 1_{\text{init}}$ uses the initial prompt; * marks the training configuration (Qwen2.5-7B, $K = 5$).

CAPER-DQN (RBS)								
Action-prior LLM	**Seen Task**				**Unseen Task**			
	$K = 1_{\text{init}}$	$K = 1$	$K = 5$	$K = 10$	$K = 1_{\text{init}}$	$K = 1$	$K = 5$	$K = 10$
Qwen2.5-3B	0.22	0.56	0.56	**0.58**	0.10	0.38	**0.58**	0.55
Qwen2.5-7B	0.22	0.67	**0.89***	0.78	0.19	0.57	**0.68**	0.58
Qwen3-8B	0.67	0.56	**0.84**	0.75	0.33	0.38	**0.69**	0.60
Llama-3.1-8B	0.33	0.56	**0.63**	0.46	0.48	0.52	**0.59**	0.50
Qwen2.5-14B	0.61	0.94	**0.99**	0.89	0.48	0.86	**0.94**	0.88
Qwen2.5-32B	0.78	0.89	**0.98**	0.95	0.67	**1.00**	0.97	0.99
Average	0.47	0.70	**0.82**	0.73	0.38	0.62	**0.74**	0.68
CAPER-CQL (RBS)								
Action-prior LLM	**Seen Task**				**Unseen Task**			
	$K = 1_{\text{init}}$	$K = 1$	$K = 5$	$K = 10$	$K = 1_{\text{init}}$	$K = 1$	$K = 5$	$K = 10$
Qwen2.5-3B	0.22	0.44	0.54	**0.59**	0.10	0.38	0.48	**0.53**
Qwen2.5-7B	0.22	0.78	**1.00***	0.89	0.19	0.48	0.63	**0.73**
Qwen3-8B	**0.67**	0.44	0.40	0.41	0.33	0.38	**0.42**	0.40
Llama-3.1-8B	0.33	0.50	**0.63**	0.51	0.48	**0.57**	0.56	0.48
Qwen2.5-14B	0.61	0.89	**0.98**	0.91	0.48	0.76	**0.97**	0.88
Qwen2.5-32B	0.78	0.83	0.99	**1.00**	0.67	**1.00**	**1.00**	0.99
Average	0.47	0.65	**0.76**	0.72	0.38	0.60	**0.68**	0.67

Table 7: Normalized returns for zero-shot TWC generalization under TAS. $K = 1_{\text{init}}$ uses the initial prompt; * marks the training configuration (Qwen2.5-7B, $K = 5$).

CAPER-DQN (TAS)								
Action-prior LLM	**Seen Task**				**Unseen Task**			
	$K = 1_{\text{init}}$	$K = 1$	$K = 5$	$K = 10$	$K = 1_{\text{init}}$	$K = 1$	$K = 5$	$K = 10$
Qwen2.5-3B	0.22	**0.56**	0.47	0.49	0.10	0.33	**0.52**	**0.52**
Qwen2.5-7B	0.22	0.83	**0.93***	0.84	0.19	**0.76**	0.70	0.66
Qwen3-8B	0.67	0.33	**0.83**	**0.83**	0.33	0.43	**0.82**	0.69
Llama-3.1-8B	0.33	**0.56**	0.54	0.44	**0.48**	**0.48**	0.42	0.38
Qwen2.5-14B	0.61	0.94	**0.98**	0.93	0.48	0.86	**0.88**	0.86
Qwen2.5-32B	0.78	**1.00**	0.97	0.98	0.67	0.90	**0.99**	0.96
Average	0.47	0.70	**0.79**	0.75	0.38	0.63	**0.72**	0.68
CAPER-CQL (TAS)								
Action-prior LLM	**Seen Task**				**Unseen Task**			
	$K = 1_{\text{init}}$	$K = 1$	$K = 5$	$K = 10$	$K = 1_{\text{init}}$	$K = 1$	$K = 5$	$K = 10$
Qwen2.5-3B	0.22	**0.61**	0.43	0.48	0.10	**0.67**	0.46	0.51
Qwen2.5-7B	0.22	0.83	**0.99***	0.80	0.19	0.67	0.66	**0.70**
Qwen3-8B	**0.67**	0.28	0.29	0.31	0.33	**0.38**	0.30	0.33
Llama-3.1-8B	0.33	0.56	**0.58**	0.46	0.48	0.57	**0.60**	0.47
Qwen2.5-14B	0.61	0.89	**0.99**	0.93	0.48	0.76	**0.93**	0.91
Qwen2.5-32B	0.78	0.89	0.92	**0.93**	0.67	**1.00**	0.95	0.96
Average	0.47	0.68	**0.70**	0.65	0.38	**0.67**	0.65	0.65

B.11 Mechanism Case Study

To examine how observed failures lead to prompt revisions, we trace one CAPER-DQN (RBS) update. We use a fixed ordering of runs and training stages and select the first event in which RBS chooses a candidate with higher validation return. This event occurs after 50 training episodes.

Failure-Guided Prior Refinement receives the three lowest-return trajectories from an independent batch of ten current-prompt rollouts, separate from task-performance evaluation.

Table 8 traces this prompt update. Failure analysis identifies incomplete delivery and actions inconsistent with the inventory or cutting-board state. Retrieval supplies the task sequence and action preconditions needed to address these errors. The selected prompt makes checks for the held item, cutting-board occupancy, and bowl completion before delivery explicit. RBS selects Candidate 3 because its validation return exceeds that of the current prompt.

Table 8: Evidence chain for one CAPER-DQN prompt update using RBS on the Salad task in *Overcooked*.

Stage	Saved evidence
Failure evidence	In low-return trajectories, the agent repeatedly attempted actions inconsistent with its inventory or the cutting-board state and sometimes tried to serve an incomplete dish.
Failure-guided diagnosis	The agent failed to assemble both chopped ingredients before delivery and entered loops involving items already held or placed on cutting boards.
Retrieved knowledge	Actions must meet inventory and cutting-board preconditions. Both ingredients must be chopped and placed in the bowl before delivery.
Prompt revision	The selected prompt made inventory and cutting-board checks explicit and reinforced the restrictions on bowl contents and delivery.
RBS selection	Candidate 3 replaced the current prompt: mean return 0.727 versus 0.413 over the respective 10-episode prompt assessments.

Prompt before the selected RBS update

In the Overcooked environment, your task is to serve a bowl containing only chopped tomato and lettuce. First, pick up the tomato and chop it at a cutting board, then pick up the lettuce and chop it at a cutting board. Ensure each ingredient is chopped before retrieving a bowl. Place both chopped ingredients into the bowl, and deliver it to the serving counter. Avoid picking up items already in hand or on cutting boards, and do not add any other ingredients.

Prompt selected by RBS

In the Overcooked environment, your task is to serve a bowl containing only chopped tomato and lettuce. First, pick up the tomato and chop it at a cutting board, then pick up the lettuce and chop it at a cutting board. Before picking up another ingredient or a bowl, verify you are not already holding it and that the cutting board is empty. Only put chopped ingredients into the bowl. Then deliver the completed bowl to the serving counter. Do not add onions or deliver an incomplete dish.

To isolate prompt revision from RL learning, we evaluate the pre-update and selected prompts on 30 random environment seeds with the same frozen DQN parameters. These trajectories are not used for training, and action selection does not query RAG. Table 9 shows that the selected prompt improves mean return and success rate while reducing episode length, invalid actions, and repeated or no-op actions.

Table 9: Frozen-policy comparison before and after one RBS prompt update over 30 evaluation seeds. Results are reported as mean $\pm$ standard error.

Prompt	Mean return ↑	Success rate ↑	Episode length ↓	Invalid-action rate ↓	Repeated/no-op actions ↓
Pre-update	0.57 ± 0.09	30.0%	28.27 ± 1.07	3.62%	8.77
Selected	$\mathbf{0.75 \pm 0.10}$	$\mathbf{46.7\%}$	$\mathbf{25.90 \pm 1.30}$	$\mathbf{2.86\%}$	$\mathbf{7.90}$

To compare behavior, we use the first paired evaluation episode in which the pre-update prompt fails and the selected prompt succeeds. Under the pre-update prompt, the agent repeats pick, place, and chop actions, assembles the bowl late, and fails within 31 decisions. With the selected prompt and

the same policy parameters, it chops both ingredients, adds them to the bowl, and serves the dish in 19 decisions. With the policy fixed, this pair illustrates the behavioral change after prompt revision.

B.12 Cost Analysis

To measure the cost of adapting the action prior, we compare training time, inference latency, and DQN-normalized return on *Overcooked (Salad)* (Table 10). Local computation for these measurements uses an NVIDIA RTX 4090D GPU. Collecting prompt data and refining prompts increase training time; RBS also spends time validating candidates. The selected task prompts increase inference latency relative to DQN-Prior. DQN's return is close to zero, so the normalized ratios are descriptive. We use absolute returns in the main experiments as the primary comparison.

After training, CAPER keeps the selected prompt fixed and uses a local Qwen model to generate action candidates. It calls DeepSeek and retrieves knowledge only during Failure-Guided Prior Refinement, not during inference-time action selection.

Table 10: Training and inference costs on *Overcooked (Salad)*. Relative return is DQN-normalized; N/A* denotes failure to learn an effective policy.

Category	Method	Selector	Inference (ms) ↓	Training (h) ↓	Relative return ↑
Online Value-Based	DQN	–	**7**	**1.08**	1.0×
	DQN-Prior	–	360	1.23	2.8×
	CAPER-DQN	RBS	445	3.25	**22.5**×
		TAS	465	2.32	10.8×
Online Policy-Based	GFlan	–	**236**	**0.33**	N/A*
	GFlan-Prior	–	268	0.40	2.4×
	CAPER-GFlan	RBS	352	0.99	**23.4**×
		TAS	327	0.69	11.9×
Fixed-Dataset CQL Backbone	CQL	–	**5**	**1.12**	1.6×
	BC	–	236	2.54	N/A*
	CQL-Prior	–	269	1.43	7.8×
	CAPER-CQL	RBS	350	2.35	**27.2**×
		TAS	312	1.99	25.1×

B.13 Knowledge-Base Composition

CAPER uses separate LightRAG indexes for Overcooked, VirtualHome, and TextWorld-Commonsense. We prepare the indexed English descriptions from public materials using the same DeepSeek model with two prompts, one for source code and one for research articles. For code, we merge the relevant source files into a Markdown document and retain the file paths in its code blocks. The code-processing prompt asks the model to describe environment initialization, observations, actions and their preconditions, state transitions, rewards, object interactions, and task variants. The paper-processing prompt extracts environment and task descriptions from the article PDF, including goal conditions and example task trajectories. Both prompts exclude learning algorithms, model architectures, optimization procedures, and experimental results. Appendix B.14.1 gives the full prompts.

We store the descriptions as UTF-8 text and index them separately for each environment. The indexes contain ten English documents. LightRAG 1.3.1 tokenizes them with tiktoken and divides them into chunks of up to 1,200 tokens with a 100-token overlap, producing the 11 chunks listed in Table 11. During prompt adaptation, Failure-Guided Prior Refinement queries the task's index and uses the retrieved evidence to generate candidate prompts. The selected prompt stays fixed until the next adaptation round; action selection uses the knowledge in that prompt without querying the index at each environment step.

Table 11: Task-specific knowledge-base composition. Indexed tokens include overlap; Goal/procedure counts documents containing goal conditions or ordered procedures.

Knowledge-base index	Documents	Chunks	Indexed tokens	Goal/procedure
Overcooked	4	4	3,130	3/2
VirtualHome	4	4	2,767	3/2
TextWorld-Commonsense	2	3	2,316	2/2
Total	**10**	**11**	**8,213**	**8/6**

Retrieval uses LightRAG's naive mode with 768-dimensional `nomic-embed-text` embeddings. LightRAG ranks at most 60 chunks, retains those above a cosine-similarity threshold of 0.2, and adds them in order within a 4,000-token text-unit budget. We use no reranker or per-document chunk cap. The query contains the current instruction, failure-analysis summary, and suggested retrieval keywords. LightRAG summarizes the retained evidence in the `retrieved_knowledge` response sent to the prompt designer. RBS and TAS use the same index construction and retrieval settings.

For the TWC transfer experiment, the index contains the six-object training task, general cleanup mechanics, and a generic seven-object example. It contains neither the exact seven-object evaluation task nor its three unique objects: red potato, red onion, and white onion. The policy and prompt therefore transfer to a different TextWorld-Commonsense task using shared domain knowledge.

B.14 Prompt Templates and Archived Examples

B.14.1 Knowledge-Base Construction Prompts

We use the following DeepSeek prompts to prepare the LightRAG indexes. The first processes the merged environment source code in Markdown with file paths; the second processes the corresponding research article. The anonymous code release includes the exact plaintext prompts and snapshots of the indexed documents.

Source-code processing prompt

```
You are a domain expert in analyzing reinforcement learning environments.

I will provide you with a merged Markdown file that contains the full source code of
     one or more simulation environments. Each code block is preceded by its absolute
     file path (e.g., # /absolute/path/to/file.py).

Your task is to read the full content and generate a comprehensive, highly detailed,
     and natural-language description of **only the environments themselves**. Do not
     include or discuss any training algorithms, reinforcement learning algorithms,
     large language models, or optimization techniques.

Focus entirely on the environments' internal logic, dynamics, and mechanics. Please
     include:
1. **What environments are implemented**
   - What are their types (e.g., grid-based, symbolic planner, 3D simulation)?
   - What tasks or domains do they simulate (e.g., cooking, household activity)?
2. **How the environment is initialized and structured**
   - How are agents, objects, and maps defined?
   - What kinds of maps or scenarios are supported?
   - What configuration files or parameters control layout and entities?
3. **What states and observations are available to the agents**
   - Are observations vector-based, symbolic, or visual (e.g., images)?
   - Are the environments partially observable? How is observability configured?
   - What information is encoded in the agent's observation (e.g., object types,
     spatial layout, internal states of items like "chopped" or "plugged in")?
4. **What actions agents can take**
   - Describe the action space in full: primitive actions, macro-actions, symbolic
     verbs, etc.
   - For each action, explain what it does, what its preconditions are, and what
     effects it has in the environment.
   - Mention any hierarchical action structures, if applicable.
5. **How the reward functions are designed**
   - What events produce positive or negative rewards?
```

```
   - Are rewards sparse or shaped? Are subtasks rewarded incrementally?
   - Include detailed breakdowns of reward computation rules if available.
6. **How objects and agents interact**
   - What are the state transitions or affordances of objects (e.g., can be chopped,
     plugged in, turned on)?
   - What interaction rules exist between agents and environment elements?
   - What happens when agents attempt valid or invalid interactions?
7. **Optional but relevant: environmental variants and parameterizations**
   - Are there different maps, modes, or difficulty levels?
   - How do those affect environment structure or agent behavior?

Your output should be:
- Written in **natural English prose**, not bullet points or schema formats.
- **Extremely detailed and specific**, with concrete examples where possible.
- Suitable for a technical reader who wants to understand everything about how the
    environment works internally.

Do not skip any important environment-specific logic, even if complex. Do not summarize
     briefly -- explain fully and precisely.

Here is the merged Markdown source file:
```

Research-article processing prompt

```
You are a domain expert in analyzing and documenting reinforcement learning (RL)
    environments based on scientific papers.
I will provide you with a PDF file of one or more research papers. Your task is to read
     the entire content and generate a precise, comprehensive, and technically detailed
     natural-language description of the simulation environments presented in the paper
    (s).
Do not describe training methods, models, reinforcement learning algorithms, or
    optimization techniques.
Focus exclusively on the internal design, logic, and mechanics of the environments
    themselves, as described in the paper.
Your output must include the following:
1. What environments are introduced or used in the paper?
What is the type of each environment? (e.g., 2D grid-world, 3D physics simulator,
    symbolic planner)
What task or domain does it simulate? (e.g., cooking, object manipulation, navigation,
    social interaction)
2. How is the environment initialized and structured?
How are agents, objects, and maps or scenes defined?
Are the environments built from handcrafted layouts, procedurally generated scenes, or
    symbolic scripts?
What parameters govern layout variation, object distribution, or task difficulty?
3. What observations are available to the agents?
Are observations symbolic, vector-based, language-based, visual (e.g., image frames),
    or multimodal?
Is the environment fully observable or partially observable?
What specific information is included in each observation (e.g., object properties,
    spatial layout, task instructions)?
4. What actions can agents take?
List and explain the entire action space (e.g., primitive actions, symbolic macro-
    actions, high-level commands)
For each action, describe:
Preconditions (what must be true for it to execute)
Effects (how it changes the state or environment)
Failure modes (what happens if action is invalid or infeasible)
5. How is reward defined?
What events trigger positive or negative reward?
Are rewards sparse (e.g., only at task completion) or shaped (e.g., with intermediate
    progress rewards)?
How is task success or failure detected in the environment?
6. How do agents and objects interact?
What affordances do objects have (e.g., can be picked up, placed, chopped, turned on)?
Are there any compositional, combinatorial, or temporal constraints (e.g., "unlock door
     before entering")?
What logic determines valid and invalid interactions?
7. Are there environment variants or difficulty settings?
Does the paper include multiple tasks, levels, maps, or scenarios per environment?
```

```
How do those variants differ in layout, complexity, instruction structure, or
     observability?
Are distractors, ambiguous language, or longer sequences used to increase difficulty?
8. Provide example trajectories in natural language
For each individual task defined in the paper's environments, extract or reconstruct at
      least one full example trajectory, written in clear and precise natural language.
Each trajectory should:
Begin with the task instruction (if applicable)
Narrate the step-by-step sequence of agent actions and observations
Describe state changes, object interactions, and decision logic at each step
Conclude with the final outcome (e.g., successful task completion, failure, and reward)
Write as if explaining to a human reader how the agent performs the task in a concrete
     episode. Use fluent prose.
Avoid bullets or tables -- for example:
"The agent starts in the kitchen and sees a tomato nearby. It walks toward the tomato,
     picks it up, and carries it to the cutting board. After chopping it, the agent
     retrieves a bowl, places the tomato inside, and finally delivers the dish to the
     serving counter."
If the paper contains diagrams, scripted programs, visual examples, or action traces,
     use them to guide your reconstruction.
Output Style and Constraints:
Use natural English prose throughout the explanation, do not use formatting such as
     tables.
All content should be in paragraph format, without complex structure.
Group content by environment, and within each environment, by task.
Only include pseudocode or formalism when necessary to clarify mechanics.
Do not describe learning algorithms, models, optimization procedures, or training
     results.
```

B.14.2 Failure-Guided Prior Refinement Templates and Prompt Examples

The templates below are taken from the Failure-Guided Prior Refinement implementation, and the examples are prompts saved during the experiments. They specify the failure-analysis input, the LightRAG query, and the instruction for generating candidate action-prior prompts.

Failure-analysis template

```
[System]
You are an expert analyst for reinforcement learning agents operating in decision-
     making environments.
Your task is to analyze some failed trajectories and identify the most likely root
     cause of the failure.

[User]
Here are the inputs you will be working with:
### Current Instruction (Used by the LLM):
{cur_instruction}

### Failed Trajectories:
{examples}
### Analysis Instructions:
1. Review the Current Instruction, including its task goal, together with the Failed
     Trajectories.
2. Identify the primary root cause(s) of the failure. Focus on:
   * Misunderstanding the task goal or current state.
   * Inefficient exploration or navigation.
   * Incorrect object interaction or use of tools.
   * Repeating errors or getting stuck in loops.
   * Missing necessary preconditions for an action.
   * Flawed high-level strategy.
3. Provide a brief and concise summary of the failure analysis (1-2 sentences maximum).
4. Suggest 1-3 keywords related to the failure cause, suitable for querying a knowledge
      base about game strategies or common pitfalls.
Return JSON:
{
  "Failure Analysis Summary": <concise summary>,
  "Suggested Keywords": <keyword1, keyword2, keyword3>
}
```

LightRAG query template

```
Current instruction is: {cur_instruction}.
Strategies related to {key_words} for preventing {failure_analysis_summary} in this
     task.
Note that you need to retrieve environment setting OR simulation environment OR task-
     related action experience NOT method NOT algorithm.
You must return JSON:
{
  "retrieved_knowledge": [
    <relevant knowledge snippet 1>,
    <relevant knowledge snippet 2>,
    "..."
  ]
}
```

Candidate-prompt generation template

```
[System]
You are an expert prompt engineer.
Your task is to optimize a prompt used to guide a Large Language Model to select
     actions towards achieving the task goal within the given environment.
The LLM agent operates in decision-making environments.
Modify the Current Instruction based on failure analysis and retrieved knowledge, while
      adhering to strict constraints.

[User]
### Current Instruction (Used by the LLM):
{cur_instruction}
### Failure Analysis:
{analysis}
### Retrieved Relevant Knowledge (From RAG):
{knowledge}

Modify the Current Instruction to create a New Instruction. Incorporate insights from
     the Failure Analysis and Retrieved Relevant Knowledge to prevent similar failures.

Constraints:
1. Incremental Modification: edit, replace, or refine existing parts; do not simply
     append large blocks.
2. Change Magnitude: remain highly semantically similar to the Current Instruction and
     keep edit distance low.
3. Clarity and Focus: remain clear, concise, and task-focused.
4. Ideally mention the target environment.

Return JSON:
{
  "new instruction": <complete new instruction text>
}
```

Representative before/after prompts. The Overcooked pair contains 16 and 136 whitespace-delimited words; the TextWorld-Commonsense pair contains 50 and 84. The optimized prompts are the strings retained by the corresponding evaluation scripts.

Overcooked (Salad): initial prompt

```
Your task is to serve the dish of a bowl only containing chopped tomato and lettuce.
```

Overcooked (Salad): archived optimized prompt

```
Your task is to serve a dish consisting solely of chopped tomato and lettuce in a bowl
within the 7 × 7 Overcooked grid-world environment. First, locate and pick up the
tomato and lettuce, ignoring distractors like onions. Navigate to separate cutting
boards to chop each ingredient (ensure you're next to the board and the ingredient is
unchopped). Fully chop each ingredient before moving to the next step--do not attempt
to serve until both are chopped. Then, retrieve a bowl, place both chopped ingredients
inside, and deliver the completed dish. Work efficiently to minimize time penalties
(-0.001 per step) while avoiding invalid actions like chopping already-chopped items or
placing unchopped ingredients in the bowl. Note your visibility is limited to a 5 × 5
grid around you. Prioritize correct sequencing: chop all required ingredients before
combining them in the bowl.
```

TextWorld-Commonsense: initial prompt

```
You find yourself in a messy house. Many things are not in their usual location.
Let's clean up this place. After you'll have done, this little house is going to be
spick and span! Look for anything that is out of place and put it away in its proper
location.
```

TextWorld-Commonsense: archived optimized prompt

```
You are in a pantry with a folding chair and a shelf. Several food items--red hot
pepper, yellow onion, peanut oil, sugar, salt, and black pepper--are misplaced on the
folding chair. Your task is to clean up by moving each item to the shelf efficiently.
Focus on one item at a time: take an item from the chair, place it on the shelf, and
proceed to the next. Avoid repeatedly moving the same item. Ensure all items are
correctly placed to organize the pantry.
```

C NOTATION

Table 12 lists the main symbols used in the method and analysis.

Table 12: Notation used throughout the paper.

Symbol	Description
RL policy and LLM action prior	
s_t, a_t, r_t	State description, selected action, and reward at time step t.
τ	Trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$.
T_j, R_j	Length and cumulative return of trajectory τ_j; equivalently, $R_j = R(\tau_j)$.
π_θ, θ	RL policy and its learnable parameters.
ϕ, ϕ_{old}, $\phi^{(i)}$	Current prompt, pre-refinement prompt, and the i-th candidate prompt, respectively.
$\pi_\phi(\cdot \mid s)$	Executable action-prior distribution induced by the LLM under prompt ϕ in state s.
K	Number of candidate actions generated by the LLM at each environment step.
N	Number of candidate prompts generated at each refinement round.
I_{adapt}	Interval between prompt-refinement opportunities.
Failure-Guided Prior Refinement	
W, F	Number of recent current-prompt trajectories considered and number of lowest-return trajectories retained as failures.
$\mathcal{D}^-$	Failure set containing the F lowest-return trajectories among the most recent W current-prompt trajectories.
p_f, $\mathcal{Q}$	Aggregate failure description and retrieval keywords produced by the problem-analysis LLM.
$\mathcal{K}$	External task-specific knowledge base.
$\mathcal{R}$	External evidence retrieved from $\mathcal{K}$ for the identified failures.
Return-Based Selection and Trajectory-Agreement Selection	
Φ	Prompt pool containing the current prompt and the N newly generated candidates.
$\bar{R}_{\text{cur}}$, $\bar{R}_{\text{prev}}$	Mean return of the current policy–prior pair and the stored reference return from the preceding RBS update.
E	Number of candidate-evaluation episodes used by RBS for each candidate prompt.
$\widehat{J}(\phi)$	Empirical environment-return score assigned to prompt ϕ by RBS.
$\mathcal{B}$	TAS trajectory source: the online training history or the fixed offline dataset.
$\mathcal{D}^+$	Set of the H highest-return trajectories selected from $\mathcal{B}$.
H	Number of high-return trajectories used by TAS.
L	Number of LLM action generations for each state–action pair and candidate prompt in TAS.
$\widehat{p}_{\phi^{(i)}}(a_{j,t} \mid s_{j,t})$	Empirical frequency with which candidate prompt $\phi^{(i)}$ reproduces action $a_{j,t}$ in state $s_{j,t}$.
ϵ	Positive numerical stabilizer used to normalize trajectory returns in TAS.
$\widetilde{R}_j$, w_j	Normalized return of trajectory τ_j and its corresponding TAS aggregation weight.
$\text{Score}(\phi)$	Return-weighted trajectory-agreement score used by TAS to rank prompts.
ϕ^*	Prompt selected by RBS or TAS for the next stage of policy learning.
Finite-sample prompt-selection analysis	
$G(\phi)$, $\widehat{G}(\phi)$	Population prompt-quality criterion and its finite-sample estimate.
$\widehat{\phi}$	Prompt that maximizes $\widehat{G}$ over the finite pool Φ.
δ	Failure probability in the uniform finite-sample guarantee.
ξ, ξ_{RBS}, ξ_{TAS}	Uniform estimation-error radius and its RBS- and TAS-specific forms.
B_R	Width of the bounded interval containing episodic returns in the RBS analysis.
$V_{H,L}$	Weighted variance proxy $\sum_{j=1}^{H} w_j^2/(T_j L)$ in the TAS error bound.
Training and evaluation	
$\mathcal{D}_{\text{off}}$	Fixed dataset used for BC and CQL policy optimization and as the TAS trajectory source in offline RL.
N_{train}	Policy-learning budget: environment episodes for online RL and optimizer steps for fixed-dataset CQL.
I_{eval}	Interval between independent task-performance evaluations, in the same units as N_{train}.
α_{DP}, β_{sel}	DQN-Prior action-selection temperature and its inverse, with $\beta_{\text{sel}} = 1/\alpha_{\text{DP}}$.
ϵ_{DQN}	Exploration probability of the online DQN ϵ_{DQN}-greedy selector.
α_{CQL}	Coefficient of the conservative CQL penalty.