BEYOND OUTCOME FEEDBACK: TRAJECTORY-QUALITY-GUIDED EXPERIENCE-BASED LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Experience-based learning enables large language model (LLM) agents to improve over time by reusing knowledge from past executions. However, existing methods typically rely on final task outcomes as feedback for experience learning, overlooking the execution process and limiting their ability to evaluate and optimize experience based on trajectory quality. To address this limitation, we propose TQG-Exp, a trajectory-quality-guided experience-based learning framework that introduces process feedback. Specifically, we introduce Trajectory Quality Loss (TQL), which jointly considers task completion and process quality to provide a learning signal beyond final task outcomes. Based on this signal, we employ paired rollouts with and without retrieved experience to guide experience optimization. Representative trajectories are selected based on their outcomes and trajectory quality to extract task-level and task-type-level experiences. Meanwhile, differences in trajectory quality between the paired groups are used to estimate the contribution of retrieved experience and update its utility. The learned utility is then combined with semantic relevance to guide subsequent experience retrieval. Experiments on ALFWorld, WebShop, and ScienceWorld with two backbone LLMs show that TQG-Exp outperforms existing experience-based methods, improving Avg.@3 over the strongest baseline by 4.50 and 8.55 percentage points on DeepSeek-V4-Flash and Qwen3.5-Flash, respectively.

1 INTRODUCTION

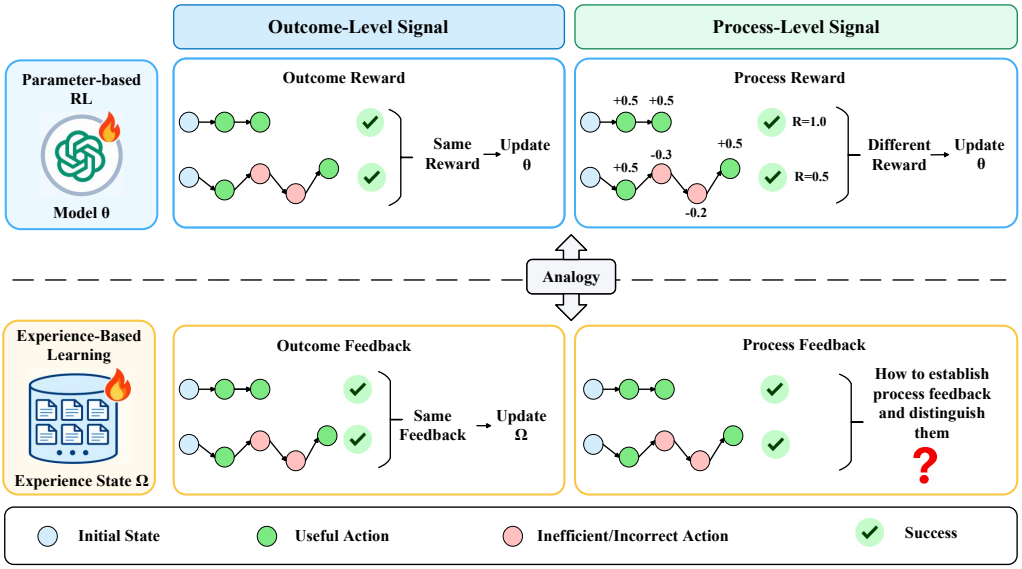


Figure 1: Illustration of outcome-level and process-level signal in reinforcement learning (top) and experience-based learning (bottom).

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.

Recent advances in large language models (LLMs) have substantially expanded the capabilities of LLM agents, enabling them to tackle complex interactive tasks such as embodied decision making, web navigation, and scientific exploration (Shinn et al., 2023; Tan et al., 2026b; Team et al., 2025). Despite these advances, many LLM agents still operate in an episodic manner, with each task typically solved independently of previous executions (Zhang et al., 2025). Consequently, useful information contained in successful and failed executions is often lost once an episode ends, forcing agents to rediscover effective strategies or repeat similar mistakes in subsequent tasks (Yin et al., 2025; Gao et al., 2025a). This limitation has motivated growing interest in experience-based learning, which enables agents to leverage experience from past executions to improve over time and transfer useful knowledge across tasks.

Existing experience-based learning methods typically evaluate experience based on final task outcomes, such as success or failure (Zhang et al., 2026; Tu et al., 2026; Feng et al., 2026; Tan et al., 2026a). However, task success does not necessarily imply high-quality experience, and task failure does not necessarily mean that the experience is entirely useless. This is because final outcomes cannot distinguish trajectories that share the same result but differ substantially in process quality. A successful trajectory may still contain redundant or invalid actions and unnecessarily long execution steps, while a failed trajectory may make substantial progress and fail only near the end. In other words, binary success or failure labels cannot adequately reflect intermediate progress and execution efficiency.

As shown in Fig. 1, experience-based learning can be viewed in analogy to parameter-based reinforcement learning: both use learning signals to optimize an adaptable component (Zhang et al., 2026). In conventional RL, reward signals guide policy optimization, typically through parameter updates. In experience-based learning, the model parameters remain fixed, while outcome feedback is used to update the external experience, determining which experiences are retained, assigned value, and reused in later decisions. In parameter-based RL, relying only on outcome rewards provides a relatively coarse learning signal and cannot distinguish different execution processes that lead to the same result. The same issue arises in experience-based learning, where the final outcome may not accurately reflect how an experience affects the execution process. Currently, some works in reinforcement learning explore process rewards to provide richer supervision for policy optimization (Wang et al., 2024b; Setlur et al., 2025; Chen et al., 2025). Following the same idea, experience learning also calls for process feedback to better evaluate and optimize external experience. This raises a key question: How can we establish process feedback to distinguish trajectory quality beyond final task outcomes?

To this end, we propose TQG-Exp, Trajectory-Quality-Guided Experience-Based Learning, which introduces process feedback into experience learning. We first introduce Trajectory Quality Loss (TQL), a process signal that jointly considers task completion and process quality, allowing trajectories with the same outcome to be distinguished by how well they are executed. We then use this signal to guide external experience optimization. Specifically, guided by TQL, paired rollouts with and without retrieved experience are used to select representative trajectories for extracting task- and task-type-level experiences and to estimate the contribution of retrieved experience based on differences in trajectory quality. The estimated utility is further combined with semantic relevance to guide subsequent experience retrieval, forming a closed loop of experience extraction, evaluation, and reuse.

In summary, our contributions are threefold:

• We formulate experience-based learning from a trajectory-quality perspective, moving beyond outcome-only feedback toward process-aware supervision for evaluating and optimizing external experience while keeping the LLM fixed.

• We propose TQG-Exp, a trajectory-quality-guided experience-based learning framework. It introduces Trajectory Quality Loss (TQL) to evaluate both task completion and process quality. Guided by TQL, paired rollouts with and without retrieved experience are used to select representative trajectories for extracting task- and task-type-level experiences and to estimate the incremental contribution of retrieved experience. The estimated utility is then combined with semantic relevance to guide subsequent experience retrieval.

• We evaluate TQG-Exp on ALFWorld, WebShop, and ScienceWorld with two base models. TQG-Exp consistently outperforms existing experience-based methods, improving Avg.@3 over the

strongest baseline by 4.50 percentage points on DeepSeek-V4-Flash and 8.55 percentage points on Qwen3.5-Flash.

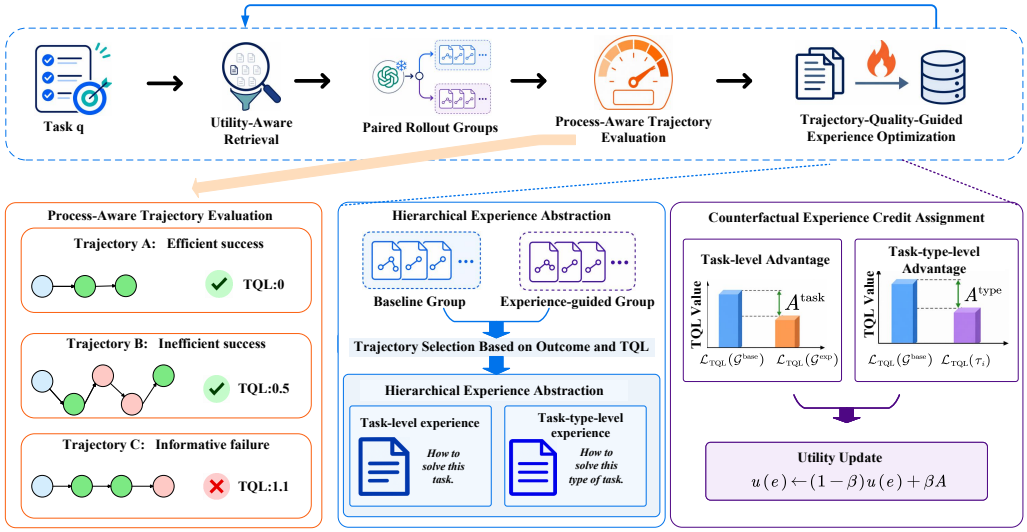


Figure 2: Overview of TQG-Exp. The top illustrates the overall experience learning loop, while the bottom panels detail process-aware trajectory evaluation, hierarchical experience abstraction, and counterfactual experience credit assignment with utility updates.

2 Related Work

LLM Agents. The rapid progress of LLMs has driven growing interest in autonomous agents that interact with external environments (Wang et al., 2024a; Wei et al., 2026). Representative frameworks such as ReAct enable agents to interleave reasoning with actions and adapt to environmental feedback (Yao et al., 2023), while AutoGen and CAMEL further support multi-agent collaboration and tool-augmented problem solving (Wu et al.; Li et al.). Despite these advances, many agents still rely on transient prompt-level context, such as recent interaction histories or task-specific hints (Dong et al., 2024). Such information is usually tied to the current episode and is not retained for future use. This is limiting in long-horizon settings, where tasks often share subgoals, action patterns, and failure modes (Zhao et al., 2024). As a result, agents may repeatedly make similar mistakes or rediscover effective strategies from scratch (Xiong et al., 2025; Wang & Chen, 2025; Hu et al., 2026). These limitations have motivated experience-based learning, where agents extract reusable knowledge from past executions and use it to improve decision making on subsequent tasks.

Experience-Based Learning for LLM Agents. Recent work has explored how LLM agents can improve from past interactions by extracting and reusing experience. Existing approaches mainly differ in whether experience learning is coupled with policy optimization or performed externally with a frozen LLM. SKILLRL (Xia et al., 2026) and D2Skill (Tu et al., 2026) jointly evolve the policy and reusable experience, while methods such as ReasoningBank (Ouyang et al., 2025), MemP (Fang et al., 2025), Training-Free GRPO (Cai et al., 2025), MemRL (Zhang et al., 2026), and DELTAMEM (Tan et al., 2026a) keep the underlying LLM fixed and improve subsequent behavior through externally stored experience. Despite differences in how experience is represented and optimized, these methods largely rely on final task outcomes or outcome-derived signals to evaluate and refine experience. Such signals provide limited information about the quality of the execution process itself. Our work instead introduces trajectory quality as the supervision signal for experience learning, allowing experience to be evaluated beyond final outcomes.

3 PRELIMINARIES

LLM Agent. Following Tu et al. (2026), we adopt the ReAct framework (Yao et al., 2022b) and formulate each task as a partially observable sequential interaction process $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T})$, where $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ denote the state, action, and observation spaces, respectively, and $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the environment transition function. Since the underlying environment state is not directly accessible, the agent makes decisions based on the task query and its interaction history.

At step t, the agent observes the context

$$h_t = (q, o_1, a_1, o_2, \ldots, a_{t-1}, o_t), \tag{1}$$

where q denotes the task query, $a_i \in \mathcal{A}$ is the action taken at step i, and $o_i \in \mathcal{O}$ is the corresponding observation. Conditioned on h_t, the frozen LLM policy π_θ generates:

$$R_t = \langle z_t, a_t \rangle \sim \pi_\theta(\cdot \mid h_t), \tag{2}$$

where z_t denotes the reasoning trace and a_t the action to execute. After a_t is executed, the environment returns the next observation o_{t+1}. The reasoning trace z_t is discarded, while (a_t, o_{t+1}) is appended to the interaction history for the next step. The interaction terminates when the agent produces a final answer y or reaches the maximum number of steps. The resulting trajectory is:

$$\tau = (q, o_1, a_1, o_2, \ldots, a_{T-1}, o_T, y). \tag{3}$$

Experience-Augmented Agent. An experience-augmented agent extends the interaction context with external experience retrieved from an experience space $\mathcal{E}$. Given the current query and interaction history, a retrieval function selects a set of relevant experiences:

$$\mathcal{E}_t^{\text{ret}} = \text{Retrieve}(q, h_t, \mathcal{E}). \tag{4}$$

The retrieved experience is incorporated into the agent context, and the response is generated as:

$$R_t = \langle z_t, a_t \rangle \sim \pi_\theta \left(\cdot \mid h_t, \mathcal{E}_t^{\text{ret}}\right), \tag{5}$$

where π_θ denotes the frozen LLM policy. The experience space can be constructed from previous trajectories or distilled experiences and provides additional context for action selection without modifying the LLM parameters.

4 METHOD

To overcome the limitations of outcome-based feedback in experience learning, we propose TQG-Exp, a new approach that integrates process-aware trajectory evaluation with trajectory-quality-guided experience optimization. As illustrated in Fig. 2, we introduce Trajectory Quality Loss (TQL) to provide process feedback beyond final task outcomes. Based on this signal, we employ paired rollouts with and without retrieved experience to select representative trajectories for experience extraction and estimate experience utility based on differences in trajectory quality. The learned utility then guides subsequent experience retrieval, forming a closed loop of experience optimization and reuse.

4.1 PROCESS-AWARE TRAJECTORY EVALUATION

We first focus on constructing a process-aware learning signal for experience learning. Process rewards in reinforcement learning provide finer-grained reward signals beyond final outcome rewards. Following this idea, we introduce process feedback into experience learning without relying on a learned reward model. Instead, we directly construct a trajectory-level quality signal from the generated execution, termed Trajectory Quality Loss (TQL).

Trajectory Quality Loss. For a trajectory τ_i, we define:

$$\mathcal{L}_{\text{TQL}}(\tau_i) = (1 - y_i) + (1 - \text{PQ}(\tau_i)), \tag{6}$$

where $y_i \in \{0, 1\}$ indicates task completion and $\text{PQ}(\tau_i)$ measures process quality based on trajectory length, invalid actions, repeated-action loops, and the position of the first error. These signals

characterize different aspects of the execution process, including efficiency, executability, behavioral stability, and progress before failure. Lower TQL indicates higher trajectory quality. The detailed definition of $\mathrm{PQ}(\tau_i)$ is provided in Appendix A.1.

Group-Level Trajectory Quality. For each task q, multiple rollouts form a group $\mathcal{G}$. We define the group-level TQL as:

$$\mathcal{L}_{\mathrm{TQL}}(\mathcal{G}) = \frac{1}{|\mathcal{G}|} \sum_{\tau_i \in \mathcal{G}} \mathcal{L}_{\mathrm{TQL}}(\tau_i) + \left(1 - \frac{1}{|\mathcal{G}|} \sum_{\tau_i \in \mathcal{G}} y_i\right). \tag{7}$$

The trajectory-level TQL evaluates the quality of an individual execution, whereas the group-level TQL summarizes the overall quality of multiple rollouts under the same task condition. In addition to the average trajectory-level TQL, the group-level formulation incorporates the group failure rate, which captures how frequently the agent fails across multiple rollouts under the same task condition.

4.2 Trajectory-Quality-Guided Experience Optimization

Having constructed the trajectory-quality signal, we next use it to optimize the external experience. We represent the adaptable experience state as:

$$\Omega = (\mathcal{E}, \mathcal{U}, \mathcal{N}), \tag{8}$$

where $\mathcal{E}$ denotes the experience space composed of stored experiences e, and $\mathcal{U} = \{u(e) \mid e \in \mathcal{E}\}$ and $\mathcal{N} = \{n(e) \mid e \in \mathcal{E}\}$ denote the utility values and retrieval counts associated with these experiences, respectively. Experience-based learning then aims to optimize this external state so that the frozen LLM produces higher-quality trajectories:

$$\Omega^\star = \arg\min_{\Omega} \mathbb{E}_{q \sim \mathcal{D},\, \tau \sim \pi_{\theta,\Omega}(\cdot|q)} \left[\mathcal{L}_{\mathrm{TQL}}(\tau)\right], \tag{9}$$

where θ remains fixed, $\pi_{\theta,\Omega}$ denotes the policy induced by the frozen LLM conditioned on the external experience state Ω, $\mathcal{D}$ denotes the task distribution, and $\mathcal{L}_{\mathrm{TQL}}(\tau)$ measures trajectory quality. We optimize the external experience state Ω through grouped rollouts under each task condition. For each task q, we perform paired rollouts consisting of a baseline group $\mathcal{G}^{\mathrm{base}}$ without retrieved experience and an experience-guided group $\mathcal{G}^{\mathrm{exp}}$ conditioned on retrieved experience. These rollout groups serve two purposes. First, trajectory quality is used to select representative trajectories from the groups for updating the experience space $\mathcal{E}$. Second, the relative quality of the paired groups is used to estimate the contribution of retrieved experience and update its utility $\mathcal{U}$. The learned utility, together with retrieval statistics $\mathcal{N}$, then guides subsequent experience retrieval. This process forms a closed loop of experience extraction, evaluation, and reuse.

Quality-Guided Hierarchical Experience Abstraction. Since the learned experience is intended to address limitations revealed during agent execution, we use baseline rollouts as the primary source of failure cases. Representative trajectories are then selected according to task outcome and trajectory-level TQL, favoring high-quality successes and contrasting them with informative failures or inefficient successes from the baseline group. The selected trajectories are distilled into two levels of experience: task-level experience captures task-specific procedures, key considerations, and failure patterns, while task-type-level experience summarizes reusable decision patterns across related tasks. Detailed extraction rules and the extraction prompt are provided in Appendix A.2 and Appendix C, respectively.

Counterfactual Experience Credit Assignment. A single TQL value cannot determine whether retrieved experience is genuinely effective, since a successful trajectory may result from the agent's inherent capabilities rather than from the retrieved experience itself. We therefore compare paired rollouts $\mathcal{G}^{\mathrm{base}}$ and $\mathcal{G}^{\mathrm{exp}}$ under the same task q to estimate the incremental contribution of retrieved experience. Task-level experience is retrieved once for each task and shared across its rollouts, whereas task-type-level experience is retrieved dynamically at each interaction step and may therefore differ across trajectories. We accordingly estimate their contributions at different granularities. For task-level experience, the contribution is measured by the difference in group-level TQL:

$$A^{\mathrm{task}}(q) = \mathcal{L}_{\mathrm{TQL}}\left(\mathcal{G}^{\mathrm{base}}\right) - \mathcal{L}_{\mathrm{TQL}}\left(\mathcal{G}^{\mathrm{exp}}\right). \tag{10}$$

For task-type-level experience, we evaluate each guided trajectory relative to the overall baseline performance:

$$A^{\text{type}}(\tau_i \mid q) = \mathcal{L}_{\text{TQL}}\left(\mathcal{G}^{\text{base}}\right) - \mathcal{L}_{\text{TQL}}(\tau_i), \qquad \tau_i \in \mathcal{G}^{\text{exp}}. \tag{11}$$

The group-level and trajectory-level TQL values are normalized to $[0, 1]$ before computing the advantages. The resulting advantages are then used to update experience utility:

$$u^l(e) \leftarrow (1 - \beta_l)u^l(e) + \beta_l A^l, \qquad l \in \{\text{task}, \text{type}\}. \tag{12}$$

A positive advantage indicates improvement over the baseline reference.

Utility-Aware Experience Retrieval. For subsequent tasks, we retrieve candidates $\mathcal{C}(q)$ based on semantic relevance, restricted to the same task type and excluding the current task's own experiences. Each candidate $e \in \mathcal{C}(q)$ is then reranked using semantic relevance and learned utility:

$$\mathcal{R}_K(q) = \text{TopK}_{e \in \mathcal{C}(q)} \left[\alpha \,\text{sim}(q, e) + (1 - \alpha) \left[u(e) + c\sqrt{\frac{\log(1 + N)}{1 + n(e)}} \right] \right], \tag{13}$$

where N is the total retrieval count, α is the relevance-utility trade-off coefficient, and c controls exploration. The top-K experiences under this combined score are selected for subsequent use. During evaluation, we also retrieve successful trajectories from the top-K_{tra} most similar tasks as additional execution exemplars. Details are provided in Appendix A.3.

5 Experiments

5.1 Setup

Datasets. Referring to the benchmark settings used in related studies (Yang et al., 2025; Shang et al., 2025; Zhang et al., 2026; Tu et al., 2026; Tan et al., 2026a), we evaluate TQG-Exp on three challenging benchmarks for LLM agents: ALFWorld (Shridhar et al., 2021), WebShop (Yao et al., 2022a), and ScienceWorld (Wang et al., 2022). Further details of the datasets are provided in Appendix B.1.

Base models. We conduct experiments with DeepSeek-V4-Flash and Qwen3.5-Flash. For each model, the same LLM serves as both the agent model for environment interaction and the reflection model for extracting experiences from the selected trajectories, without requiring an additional LLM.

Baseline Methods. We compare our method with five baselines: No Experience, ReasoningBank (Ouyang et al., 2025), MemP (Fang et al., 2025), MemRL (Zhang et al., 2026), and DeltaMem (Tan et al., 2026a). Implementation details and configuration settings are provided in Appendix B.2.

Evaluation Metrics. We report Average@3 and Pass@3. Average@3 is the average success rate across three independent rollouts, while Pass@3 is the percentage of tasks solved in at least one of the three rollouts.

5.2 Main results

Table 1: Performance comparison across ALFWorld, WebShop, and ScienceWorld. Best results are in **bold**, and second-best results are underlined.

Model	Method	ALFWorld				WebShop		ScienceWorld		Avg.	
		Seen		Unseen							
		Avg.@3 ↑	Pass@3 ↑	Avg.@3 ↑	Pass@3 ↑	Avg.@3 ↑	Pass@3 ↑	Avg.@3 ↑	Pass@3 ↑	Avg.@3 ↑	Pass@3 ↑
DeepSeek-V4-Flash	No Experience	68.57	76.43	74.88	85.82	42.33	52.00	47.00	60.00	58.20	68.56
	ReasoningBank (Ouyang et al., 2025)	75.24	82.14	75.37	85.07	39.67	48.00	48.33	55.00	59.65	67.55
	MemP (Fang et al., 2025)	84.76	91.43	88.06	94.03	36.33	48.00	57.33	63.00	66.62	74.12
	MemRL (Zhang et al., 2026)	90.00	92.86	90.30	93.28	73.67	78.00	69.33	76.00	80.83	85.04
	DELTAMEM (Tan et al., 2026a)	80.95	87.14	79.85	85.82	48.33	60.00	56.67	65.00	66.45	74.49
	TQG-Exp	**96.19**	**96.43**	**90.80**	**94.78**	**79.67**	**85.00**	**74.67**	**81.00**	**85.33**	**89.30**
Qwen3.5-Flash	No Experience	65.00	75.00	70.40	79.10	51.67	67.00	42.00	55.00	57.27	69.03
	ReasoningBank (Ouyang et al., 2025)	73.10	80.00	79.60	88.06	49.33	61.00	46.67	57.00	62.18	71.52
	MemP (Fang et al., 2025)	79.52	87.86	85.32	91.04	60.33	73.00	58.67	73.00	70.96	81.23
	MemRL (Zhang et al., 2026)	87.14	89.29	85.82	89.55	75.00	79.00	57.67	70.00	76.41	81.96
	DELTAMEM (Tan et al., 2026a)	72.62	79.29	76.37	85.82	60.00	72.00	54.67	68.00	65.92	76.28
	TQG-Exp	**93.10**	**96.43**	**89.05**	**93.28**	**80.00**	**82.00**	**77.67**	**94.00**	**84.96**	**91.43**

Tab. 1 compares TQG-Exp with existing experience-based methods across three interactive benchmarks and two backbone LLMs. TQG-Exp achieves the best Average@3 and Pass@3 in all evaluated settings. Among the baselines, MemRL achieves the strongest average performance across the evaluated settings. It stores complete trajectories as experiences and updates their utility through runtime reinforcement learning, enabling retrieval based on both semantic relevance and learned utility. Nevertheless, TQG-Exp consistently surpasses MemRL with both backbone LLMs. With DeepSeek-V4-Flash, the overall Average@3 increases from 80.83 to 85.33 and Pass@3 from 85.04 to 89.30. With Qwen3.5-Flash, Average@3 improves from 76.41 to 84.96 and Pass@3 from 81.96 to 91.43.

The advantage over MemRL is also consistent at the benchmark level. On ALFWorld, TQG-Exp outperforms MemRL on both seen and unseen splits with both backbone models, suggesting that the learned experience remains useful on unseen task instances. The gains extend to WebShop and ScienceWorld. In particular, Average@3 improves over MemRL by 6.00 points on WebShop with DeepSeek-V4-Flash and by 20.00 points on ScienceWorld with Qwen3.5-Flash. Taken together, these results highlight the value of process feedback in experience learning, where TQL provides a finer-grained learning signal beyond task outcomes.

5.3 Performance and Interaction Efficiency

Table 2: Performance and efficiency on ALFWorld with DeepSeek-V4-Flash.

Split	Method	Performance & Efficiency			
		Avg.@3 ↑	Avg. Steps ↓	Tok./Ep. ↓	Tok./Succ. ↓
Seen	No Experience	68.57	18.17	**27.11k**	40.10k
	MemRL	90.00	11.13	114.34k	127.04k
	TQG-Exp	**96.19**	**10.27**	38.06k	**39.57k**
Unseen	No Experience	74.88	18.19	**27.89k**	**37.25k**
	MemRL	90.30	12.31	125.28k	138.73k
	TQG-Exp	**90.80**	**12.24**	48.76k	53.70k

Among the experience-based learning baselines, MemRL achieves the strongest overall performance, and we therefore use it as the primary reference for the efficiency analysis. As shown in Tab. 2, TQG-Exp improves Avg.@3 from 90.00 to 96.19 on seen tasks and from 90.30 to 90.80 on unseen tasks. These gains are achieved with fewer interaction steps, reducing the average episode length from 11.13 to 10.27 on seen tasks and from 12.31 to 12.24 on unseen tasks. The reduction in token consumption is more substantial. On seen tasks, TQG-Exp reduces Tokens/Episode from 114.34k to 38.06k and Tokens/Success from 127.04k to 39.57k, corresponding to reductions of 66.7% and 68.9%, respectively. On unseen tasks, both metrics are reduced by approximately 61%. Overall, TQG-Exp achieves higher task performance than MemRL while requiring fewer interaction steps and substantially fewer tokens. We additionally report the corresponding efficiency results on the other benchmarks in Appendix B.3.

5.4 Ablation Studies and Analysis

Table 3: Ablation on ALFWorld seen and unseen tasks. All results are Avg.@3. Hier. Exp., Util. Ret., and Traj. Ret. denote Hierarchical Experience, Utility-Aware Experience Retrieval, and Trajectory Retrieval.

Model	Components				ALFWorld	
	Hier. Exp.	Util. Ret.	TQL	Traj. Ret.	Seen	Unseen
DeepSeek-V4-Flash	✓	✓	✓	✓	**96.19**	**90.80**
	✓	✓	✓	✗	91.43	89.80
	✓	✓	✗	✗	78.81	79.35
	✓	✗	✗	✗	84.76	83.08
	✗	✓	✓	✓	93.57	89.55
	✗	✗	✗	✗	68.57	74.88
Qwen3.5-Flash	✓	✓	✓	✓	**93.10**	**89.05**
	✓	✓	✓	✗	88.10	86.32
	✓	✓	✗	✗	76.90	82.34
	✓	✗	✗	✗	75.95	78.11
	✗	✓	✓	✓	88.60	85.57
	✗	✗	✗	✗	65.00	70.40

Both Utility-Aware Experience Retrieval and Trajectory Retrieval are built on the trajectory-quality signal provided by TQL: the former uses TQL to learn experience utility, while the latter uses TQL to select high-quality successful trajectories. We therefore conduct a dependency-aware progressive ablation over these three components to examine their respective contributions. Specifically, we first remove Trajectory Retrieval, disabling successful trajectory exemplars. We then replace TQL with the outcome-only signal $1 - y$, and finally remove Utility-Aware Experience Retrieval, reducing experience retrieval to cosine similarity alone. Hierarchical Experience is ablated separately by removing task-type-level experiences while retaining task-level experiences.

As shown in Tab. 3, removing Trajectory Retrieval consistently reduces performance, while replacing TQL with outcome-only feedback leads to a larger degradation, highlighting the importance of TQL for experience learning. The effect of Utility-Aware Experience Retrieval further depends on the quality of its utility signal: Qwen3.5-Flash degrades further under cosine-only retrieval, whereas DeepSeek-V4-Flash partially recovers when the outcome-based utility is no longer used for retrieval. This suggests that, for DeepSeek-V4-Flash, utility estimated from the coarse $1 - y$ signal may introduce noise into Utility-Aware Experience Retrieval. This difference may stem from the stronger base capability of DeepSeek-V4-Flash, for which the binary $1 - y$ signal becomes less discriminative because many experiences lead to successful outcomes. As a result, cosine similarity can be more reliable than retrieval with coarse utility estimates, while outcome-based utility remains more informative for Qwen3.5-Flash. Separately removing task-type-level experience also reduces performance across both models, showing that reusable local decision patterns complement task-specific global planning. Overall, the full model achieves the strongest performance through the combined use of TQL, Utility-Aware Experience Retrieval, Trajectory Retrieval, and Hierarchical Experience.

5.5 Effect of relevance-utility trade-off coefficient α

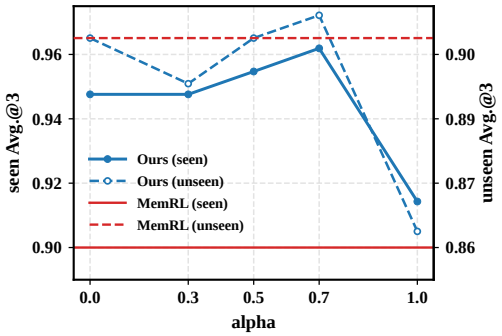


Figure 3: Sensitivity of performance to α on ALFWorld using DeepSeek-V4-Flash.

Fig. 3 examines the sensitivity of our method to the relevance-utility trade-off coefficient α on both seen and unseen tasks. Performance remains relatively stable for $\alpha \in [0, 0.7]$, with both splits achieving their highest performance at $\alpha = 0.7$, where our method outperforms MemRL on both seen and unseen tasks. However, performance drops noticeably at $\alpha = 1$, suggesting that learned utility provides useful information beyond semantic relevance. These results indicate that combining semantic relevance with learned utility improves experience retrieval, with $\alpha = 0.7$ providing the highest observed performance on both splits. Although performance declines at $\alpha = 1$, our method maintains relatively high performance, particularly on seen tasks, where it still outperforms MemRL. At $\alpha = 1$, TQL-guided experience extraction and trajectory selection, along with task-type-based candidate filtering, remain unchanged, which may help maintain relatively high performance.

5.6 Epoch-wise Comparison of Trajectory Quality Loss and Task Success Rate

We further examine the evolution of trajectory quality and task performance during experience learning using the epoch-wise TQL sum and task success rate. The epoch-wise TQL sum is defined as the sum of the original trajectory-level TQL values over all rollouts in the experience-guided groups collected in each epoch, while the success rate is the average success rate of these rollouts. To examine the effect of trajectory-quality feedback, we construct an outcome-only variant in which TQL is replaced by $1 - y$, where y indicates task success. Importantly, $1 - y$ is used only as the learning signal, and both variants are evaluated using the same original TQL metric to ensure a consistent comparison.

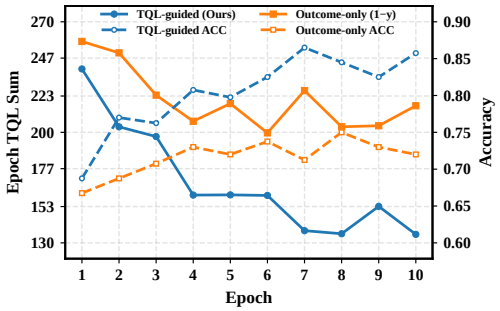


Figure 4: Epoch-wise TQL sum and task success rate during experience learning. Solid lines indicate TQL sums (left axis), while dashed lines indicate task success rates (right axis).

As shown in Fig. 4, the TQL-guided variant exhibits a clearer overall decrease in epoch-wise TQL sum, accompanied by an increasing trend in task success rate. In contrast, the outcome-only variant shows a smaller improvement in success rate, while its TQL sum decreases less consistently and exhibits larger fluctuations in later epochs. These results suggest that trajectory-quality feedback provides more informative guidance for expe-

rience optimization than outcome-only feedback, leading to improvements in both overall trajectory quality and task success. Additional epoch-wise TQL-sum results on other benchmarks are provided in Appendix B.4.

5.7 Semantic Evolution of Hierarchical Experience

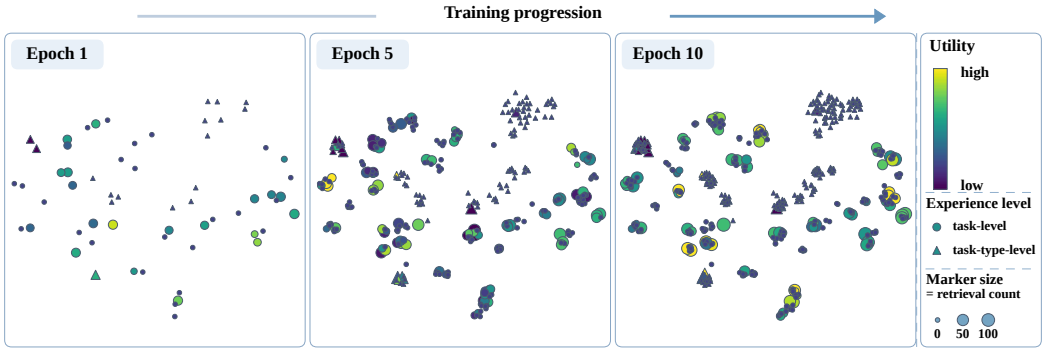


Figure 5: Semantic evolution of the ALFWorld experience bank. Circles denote task experiences, triangles denote task-type experiences, color indicates utility, and marker size indicates retrieval count.

Fig.5 visualizes how the hierarchical experience space evolves over training on ALFWorld with DeepSeek-V4-Flash as the backbone LLM. We encode the complete textual content of each task-level and task-type-level experience using Qwen3-Embedding-0.6B and jointly project the experience-bank snapshots from Epochs 1, 5, and 10 into a shared two-dimensional t-SNE space. Utility and raw retrieval count determine marker color and size, respectively.

The projection reveals clear but distinct clustering structures at the two experience levels. Task-level experiences form multiple localized clusters distributed across the semantic space, reflecting diverse task-specific goals and procedures. Task-type-level experiences also exhibit compact clusters, but tend to concentrate in more distinct semantic regions, consistent with their role in capturing reusable stage-level decision patterns. The different spatial organizations suggest that the two levels encode complementary knowledge at different levels of abstraction. As training progresses, experiences become increasingly dense within established semantic regions, indicating that the experience space gradually shifts from expanding coverage to refining previously acquired knowledge. Meanwhile, the number of high-utility and frequently retrieved experiences grows, as shown by the increasing prevalence of markers associated with higher utility values and larger marker sizes in later snapshots. Additionally, we analyze the evolution of experience bank sizes during training in Appendix B.5.

6 Conclusion

In this work, we proposed TQG-Exp, a trajectory-quality-guided experience-based learning framework that moves beyond outcome-based feedback by considering the quality of the execution process. We introduced Trajectory Quality Loss (TQL) to evaluate both task completion and process quality, providing a learning signal for experience optimization. Using this signal, we employed paired rollouts with and without retrieved experience to select representative trajectories for hierarchical experience extraction and estimate the contribution of retrieved experience. The estimated utility was then incorporated into subsequent experience retrieval. Experiments on ALFWorld, WebShop, and ScienceWorld showed that TQG-Exp consistently outperformed existing experience-based methods, improving overall Avg.@3 over the strongest baseline by 4.50 and 8.55 percentage points on DeepSeek-V4-Flash and Qwen3.5-Flash, respectively. These results highlight the potential of process feedback for improving experience learning.

AI USE STATEMENT

Generative AI tools were used to assist with language editing and polishing, including improvements to grammar, clarity, and readability. All AI-assisted revisions were carefully reviewed and revised where necessary by the authors to ensure that they accurately reflect the intended scientific meaning. The authors take full responsibility for the content of the final manuscript.

REFERENCES

Yuzheng Cai, Siqi Cai, Yuchen Shi, Zihan Xu, Lichao Chen, Yulei Qin, Xiaoyu Tan, Gang Li, Zongyi Li, Haojia Lin, et al. Training-free group relative policy optimization. *arXiv preprint arXiv:2510.08191*, 2025.

Hongzhan Chen, Tao Yang, Shiping Gao, Ruijun Chen, Xiaojun Quan, Hongtao Tian, and Ting Yao. Discriminative policy optimization for token-level reward models. *arXiv preprint arXiv:2505.23363*, 2025.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, et al. A survey on in-context learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 1107–1128, 2024.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. *arXiv e-prints*, pp. arXiv–2508, 2025.

Tao Feng, Chongrui Ye, Tianyang Luo, Jingjun Xu, Xueqiang Xu, Haozhen Zhang, Zhigang Hua, Yan Xie, Shuang Yang, Ge Liu, et al. Expgraph: Model-agnostic experience learning with graph-structured memory for llm agents. *arXiv preprint arXiv:2605.30712*, 2026.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025a.

Pengyu Gao, Jinming Zhao, Xinyue Chen, and Long Yilin. An efficient context-dependent memory framework for llm-centric agents. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 1055–1069, 2025b.

Qisheng Hu, Quanyu Long, and Wenya Wang. When continual learning moves to memory: A study of experience reuse in llm agents. *arXiv preprint arXiv:2604.27003*, 2026.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 60808–60838, 2025.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pp. 8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Haoran Tan, Zeyu Zhang, Zhicheng Cao, Rui Li, and Xu Chen. Deltamem: Incremental experience memory for llm agents via residual trees. *arXiv preprint arXiv:2606.03083*, 2026a.

Haoran Tan, Zeyu Zhang, Chen Ma, Tianze Liu, Quanyu Dai, and Xu Chen. From coarse to fine: Self-adaptive hierarchical planning for llm agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 1554–1569, 2026b.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, et al. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*, 2025.

Songjun Tu, Chengdong Xu, Qichao Zhang, Yaocheng Zhang, Xiangyuan Lan, Linjing Li, and Dongbin Zhao. Dynamic dual-granularity skill bank for agentic rl. *arXiv preprint arXiv:2603.28716*, 2026.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024a.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024b.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Yu Wang and Xi Chen. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*, 2025.

Tianxin Wei, Ting-Wei Li, Zhining Liu, Xuying Ning, Ze Yang, Jiaru Zou, Zhichen Zeng, Ruizhong Qiu, Xiao Lin, Dongqi Fu, et al. Agentic reasoning for large language models. *arXiv preprint arXiv:2601.12538*, 2026.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

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

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts llm agents: An empirical study of experience-following behavior. *arXiv preprint arXiv:2505.16067*, 2025.

Wei Yang, Jinwei Xiao, Hongming Zhang, Qingyang Zhang, Yanna Wang, and Bo Xu. Coarse-to-fine grounded memory for llm agent planning. *arXiv preprint arXiv:2508.15305*, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757, 2022a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Shaozhe Yin, Jinyu Guo, Kai Shuang, Xia Liu, and Ruize Ou. Learning wisdom from errors: Promoting llm's continual relation learning through exploiting error cases. *arXiv preprint arXiv:2508.12031*, 2025.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*, 2025.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A Additional Method Details

A.1 Detailed Definition of Trajectory Quality Loss

We define Trajectory Quality Loss (TQL) to evaluate the quality of an interaction trajectory by jointly considering task completion and process quality. For a trajectory τ_i corresponding to task q, let $y_i \in \{0, 1\}$ indicate whether the task is successfully completed. The trajectory-level TQL is defined as:

$$\mathcal{L}_{\mathrm{TQL}}(\tau_i) = (1 - y_i) + (1 - \mathrm{PQ}(\tau_i)), \tag{14}$$

where $\mathrm{PQ}(\tau_i) \in (0, 1]$ denotes the quality of the execution process. A lower TQL value indicates a higher-quality trajectory.

We measure process quality from four aspects: trajectory length, action validity, repeated-action loops, and the position of the first detected error. For each trajectory τ_i, let T_i denote the number of interaction steps. Let μ_T and σ_T denote the mean and standard deviation of the lengths of all trajectories sampled for task q. We first define the positive length deviation as:

$$z_i^+ = \max\left(\frac{T_i - \mu_T}{\sigma_T}, 0\right), \tag{15}$$

where σ_T is set to 1 when the empirical standard deviation is zero. The resulting deviation is then normalized over all trajectories of the same task q to obtain the length penalty:

$$\ell_i = \begin{cases} \dfrac{z_i^+}{\max_j z_j^+}, & \text{if } \max_j z_j^+ > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

We next measure action validity. Let $v_{i,t} \in \{0, 1\}$ indicate whether the action at step t can be parsed and accepted as a valid executable action by the corresponding environment. The action-validity ratio is defined as:

$$r_i^{\mathrm{valid}} = \frac{1}{T_i} \sum_{t=1}^{T_i} v_{i,t}. \tag{17}$$

To account for repetitive behavior, we further introduce a repeated-action loop penalty. We define a repeated-action loop as an immediately repeated contiguous block of actions. Specifically, for a

period $p \in \{1, \ldots, P\}$ and a start position s, the action subsequence $(a_{i,s}, a_{i,s+1}, \ldots, a_{i,s+p-1})$ is considered to form a repeated-action loop if it is immediately followed by an identical subsequence:

$$(a_{i,s}, a_{i,s+1}, \ldots, a_{i,s+p-1}) = (a_{i,s+p}, a_{i,s+p+1}, \ldots, a_{i,s+2p-1}). \tag{18}$$

Here, p denotes the loop period, and P is the maximum period considered. For example, period-1 loops capture repetitions such as "open fridge, open fridge", while period-2 loops capture alternating patterns such as "go north, go south, go north, go south". When such a match is found, we mark only the later copy of the repeated block, $\{s+p, \ldots, s+2p-1\}$, as loop positions. Since loop positions are represented as a set over all detected matches across different starts s and periods $p \leq P$, an action position marked multiple times contributes only once to n_i^{loop}.

Let n_i^{loop} denote the number of unique action positions in trajectory τ_i that are marked as belonging to such immediately repeated blocks. The loop penalty is defined as:

$$r_i^{\text{loop}} = \frac{n_i^{\text{loop}}}{\max(T_i - 1, 1)}. \tag{19}$$

We also consider when the first error occurs. Let t_i^{err} denote the first step at which an error is detected. A step is regarded as erroneous if it contains an invalid action, an observation indicating that nothing happens, or an action belonging to a repeated-action loop. The early-error penalty is defined as:

$$p_i^{\text{err}} = \begin{cases} 0, & \text{if no error is detected,} \\ 1 - \dfrac{t_i^{\text{err}}}{T_i}, & \text{otherwise.} \end{cases} \tag{20}$$

This assigns a larger penalty to errors that occur earlier in the trajectory. The process quality score is then computed as:

$$\text{PQ}(\tau_i) = \exp\left(-\ell_i - \left(1 - r_i^{\text{valid}}\right) - r_i^{\text{loop}} - p_i^{\text{err}}\right), \tag{21}$$

These four components capture complementary and environment-agnostic aspects of interactive process quality: efficiency (trajectory length), executability (action validity), behavioral stability(repeated-action loops), and progress before failure (first-error position). Importantly, they are computed directly from environment interactions without requiring an additional learned reward-model or task-specific process annotations.

A.2 Experience Extraction

A.2.1 Task-Level Experience Extraction

For each task q, the paired rollouts produce an experience-guided group $\mathcal{G}^{\text{exp}}$ and a baseline group $\mathcal{G}^{\text{base}}$. Reflecting on all sampled trajectories is often unnecessary and may introduce redundant evidence. We therefore select a small set of representative trajectories that reveal either effective behavior or informative mistakes. Let:

$$\mathcal{G}^{all} = \mathcal{G}^{\text{exp}} \cup \mathcal{G}^{\text{base}} \tag{22}$$

denote all sampled trajectories for the task. Each trajectory τ_i has a success indicator y_i and a trajectory-level TQL value $\mathcal{L}_{\text{TQL}}(\tau_i)$, where lower TQL indicates a better trajectory.

We first collect all successful trajectories:

$$\mathcal{G}^{+} = \left\{\tau_i \in \mathcal{G}^{all} : y_i = 1\right\}. \tag{23}$$

We also divide the baseline trajectories into successful and failed subsets:

$$\mathcal{G}^{\text{base},+} = \left\{\tau_i \in \mathcal{G}^{\text{base}} : y_i = 1\right\}, \qquad \mathcal{G}^{\text{base},-} = \left\{\tau_i \in \mathcal{G}^{\text{base}} : y_i = 0\right\}. \tag{24}$$

We draw failure representatives from the baseline group because these failures show what the agent cannot reliably handle without experience support. Distilling such failures helps the experience capture mistakes that future retrieved experiences should prevent. In contrast, the best successful representatives may come from either group, since both baseline and experience-guided successes can provide evidence of effective behavior.

When the corresponding sets are non-empty, we choose representatives according to success status and TQL. Among all successful trajectories, we select the cleanest successful solution:

$$\tau_{\text{succ}}^{\text{best}} = \arg\min_{\tau_i \in \mathcal{G}^+} \mathcal{L}_{\text{TQL}}(\tau_i). \tag{25}$$

Among failed baseline trajectories, we select two failures with different degrees of quality:

$$\tau_{\text{fail}}^{\text{near}} = \arg\min_{\tau_i \in \mathcal{G}^{\text{base},-}} \mathcal{L}_{\text{TQL}}(\tau_i), \qquad \tau_{\text{fail}}^{\text{bad}} = \arg\max_{\tau_i \in \mathcal{G}^{\text{base},-}} \mathcal{L}_{\text{TQL}}(\tau_i). \tag{26}$$

The first is the least severe failed attempt and may contain partial progress, whereas the second captures a more severe failure pattern. If baseline trajectories all succeed, failures are unavailable; in this case, we select the least efficient successful baseline trajectory:

$$\tau_{\text{succ}}^{\text{worst}} = \arg\max_{\tau_i \in \mathcal{G}^{\text{base},+}} \mathcal{L}_{TQL}(\tau_i). \tag{27}$$

These representatives are used according to the rollout outcome. For task q, we apply an ordered decision rule to its paired rollout group $\mathcal{G}^{all} = \mathcal{G}^{\text{exp}} \cup \mathcal{G}^{\text{base}}$. The rule selects at most one reflection mode. If $\mathcal{G}^{all}$ contains at least one successful trajectory and the baseline group $\mathcal{G}^{\text{base}}$ contains at least one failed trajectory, we use failure-aware reflection. If no trajectory in $\mathcal{G}^{all}$ succeeds but $\mathcal{G}^{\text{base}}$ contains failed trajectories, we use failure-only reflection. If all available baseline trajectories succeed, we consider success-contrast reflection only when the baseline group-level TQL exceeds a threshold. In all remaining cases, we skip reflection because the sampled rollouts provide insufficient new evidence for experience extraction.

Case 1: Failure-aware reflection. This case is selected when $\mathcal{G}^+ \neq \emptyset$ and $\mathcal{G}^{\text{base},-} \neq \emptyset$, where $\mathcal{G}^+$ contains successful trajectories from either the experience-guided or baseline group. We compare the best successful trajectory $\tau_{\text{succ}}^{\text{best}}$ with the near-miss and severe baseline failures, $\tau_{\text{fail}}^{\text{near}}$ and $\tau_{\text{fail}}^{\text{bad}}$. The successful trajectory reveals effective task-solving steps, the near-miss failure can help uncover overlooked checks or incomplete actions, and the severe failure can expose serious error patterns to avoid.

Case 2: Failure-only reflection. This case is selected when $\mathcal{G}^+ = \emptyset$ and $\mathcal{G}^{\text{base},-} \neq \emptyset$. We compare $\tau_{\text{fail}}^{\text{near}}$ and $\tau_{\text{fail}}^{\text{bad}}$. Although no successful solution is observed, this comparison can still reveal partial progress, missing checks, and recurring error patterns by contrasting a less severe failure with a more severe one.

Case 3: Success-contrast reflection. This case is considered when $\mathcal{G}^{\text{base},-} = \emptyset$ and $\mathcal{G}^{\text{base},+} \neq \emptyset$. Because all baseline trajectories succeed, reflection is performed only when their behavior remains inefficient or unstable. We quantify this using the baseline group-level TQL:

$$\mathcal{L}_{\text{TQL}}(\mathcal{G}^{base}) = \frac{1}{|\mathcal{G}^{\text{base}}|} \sum_{\tau_i \in \mathcal{G}^{\text{base}}} \mathcal{L}_{TQL}(\tau_i) + \left(1 - \frac{1}{|\mathcal{G}^{\text{base}}|} \sum_{\tau_i \in \mathcal{G}^{\text{base}}} y_i\right). \tag{28}$$

If $\mathcal{L}_{\text{TQL}}(\mathcal{G}^{base}) \geq 0.7$, we compare $\tau_{\text{succ}}^{\text{best}}$ with $\tau_{\text{succ}}^{\text{worst}}$ to extract lessons about efficient action ordering, avoiding redundant actions, and stabilizing successful behavior. Otherwise, we skip reflection because the baseline successes already provide little new information.

For each task q, the selected representatives, when available, are passed to the reflection model, yielding at most one reflection request for the paired rollout group $\mathcal{G} = \mathcal{G}^{\text{exp}} \cup \mathcal{G}^{\text{base}}$. The model distills the representatives into one to three ordered task-stage experiences. Each stage specifies when the lesson should be applied, the principle to follow, and, when applicable, the failure pattern it is intended to prevent. The resulting task-level experience is stored in $\mathcal{E}^{\text{task}}$ and buffered according to task type for later abstraction into task-type-level experience.

A.2.2 Task-Type Experience Distillation

Task-level experiences summarize the solution patterns observed in individual task instances. However, they may contain instance-specific details that limit their transferability. Since tasks of the same type often share common stages, decision patterns, and failure modes, we periodically distill task-level experiences into reusable task-type experiences.

Let $\phi(q)$ denote the task type of task q. For each task type z, we maintain a temporary buffer

$$\mathcal{B}_z^{\text{task}} = (\mathcal{E}_z, \mathcal{Q}_z), \tag{29}$$

where $\mathcal{E}_z$ contains valid task-level experiences extracted from tasks satisfying $\phi(q) = z$, and $\mathcal{Q}_z$ contains their corresponding task-level TQL values. An experience is added to the buffer only when task-level reflection produces a valid result.

The buffer is processed once it contains enough distinct task instances or accumulates sufficient TQL evidence:

$$|\text{UniqueTasks}(\mathcal{E}_z)| \geq M_{\min} \quad \vee \sum_{q \in \text{UniqueTasks}(\mathcal{E}_z)} \text{TQL}_q > \delta_{\text{type}}. \tag{30}$$

After processing, the buffer is cleared and a new buffer is created for later tasks. In our implementation, $M_{\min}$ and δ_{type} are configurable and are set to 4 and 5.0 by default.

Given the buffered staged task-level experiences and their associated TQL values, a reflection model extracts common and reusable patterns across tasks of the same type. The model operates only on the provided task-level experiences and does not directly access the raw trajectories. It merges semantically similar stages even when their wording, objects, products, or environment details differ, while avoiding the reproduction of a single concrete task unless it expresses a reusable task-type pattern.

For each processed buffer, the reflection model generates up to $K_{\max}$ candidate experiences:

$$\mathcal{C}_z = \left\{ e_{z,1}^{\text{type}}, \ldots, e_{z,K_z}^{\text{type}} \right\}, \qquad 1 \leq K_z \leq K_{\max}, \tag{31}$$

where $K_{\max} = 3$ by default. Candidates with duplicate content are removed before being inserted into memory.

Each remaining candidate is then compared with the existing task-type experiences of type z. Based on the candidate and the existing memory, the same model used for trajectory execution selects one of four operations: add, merge, modify, or discard.

Add. A candidate is added when it introduces a new task subtype, stage, ordering pattern, critical step, or failure mode that is not covered by the existing memory:

$$\mathcal{E}_z^{\text{type}} \leftarrow \mathcal{E}_z^{\text{type}} \cup \left\{ e^{\text{type}} \right\}. \tag{32}$$

If no task-type experience of type z exists, the candidate is added directly.

Merge. A candidate is merged with an existing experience when both express the same main principle but the candidate provides additional useful details, such as new critical steps, failure patterns, or avoidance rules:

$$e_j^{\text{type}} \leftarrow \text{Merge}\left(e_j^{\text{type}}, e^{\text{type}} \right). \tag{33}$$

Modify. A candidate is used to revise an existing experience when the current experience is misleading, overly broad, overly narrow, outdated, or inconsistent with the buffered evidence:

$$e_j^{\text{type}} \leftarrow \text{Modify}\left(e_j^{\text{type}}, e^{\text{type}} \right). \tag{34}$$

The revised experience retains the same memory identifier, increments its version, and decays its utility to reflect the uncertainty introduced by the update.

Discard. A candidate is discarded when it is redundant, too generic, or does not provide actionable information beyond the existing memory:

$$\mathcal{E}_z^{\text{type}} \leftarrow \mathcal{E}_z^{\text{type}}. \tag{35}$$

A.3 Trajectory Collection and Retrieval

We construct a trajectory pool from the training tasks for use as execution exemplars during evaluation. The procedure starts from trajectories retained at the final training epoch, followed by an additional collection stage and trajectory update.

Initial Trajectory Selection. For each training task q, we retain the successful trajectory with the minimum TQL from the final training epoch:

$$\tau_q^{\mathrm{ret}} = \arg \min_{\tau \in \mathcal{G}_q^{\mathrm{final}},\, y(\tau)=1} \mathcal{L}_{\mathrm{TQL}}(\tau), \tag{36}$$

where $\mathcal{G}_q^{\mathrm{final}}$ denotes the trajectories generated for q at the final training epoch. These trajectories form the initial trajectory pool.

Trajectory Collection. We then perform an additional collection stage over the training tasks. For each task q, the agent retrieves learned experiences together with previously retained successful trajectories from similar tasks and uses them as execution context. Among the successful trajectories generated in this stage, the one with the minimum TQL is selected as the current candidate:

$$\tau_q^{\mathrm{cand}} = \arg \min_{\tau \in \mathcal{G}_q^{\mathrm{col}},\, y(\tau)=1} \mathcal{L}_{\mathrm{TQL}}(\tau), \tag{37}$$

where $\mathcal{G}_q^{\mathrm{col}}$ denotes the trajectories generated during trajectory collection.

Trajectory Update. For each training task q, the current candidate τ_q^{cand} is compared with the previously retained trajectory τ_q^{ret}. Since both are successful, we compare them lexicographically by episode length, invalid actions, and corrected actions.

$$\tau_q^* = \mathrm{Better}\left(\tau_q^{\mathrm{ret}}, \tau_q^{\mathrm{cand}}\right). \tag{38}$$

Applying this procedure to all training tasks yields the final trajectory pool.

Trajectory Retrieval. During evaluation, we rank training tasks by semantic similarity to the current task and retrieve successful trajectories from the top-K_{tra} most similar tasks as execution exemplars. Trajectory retrieval is based only on task similarity and is separate from the Utility-Aware Experience Retrieval described in Section 4.2.

B Experiment Details

B.1 Dataset Composition

We evaluate TQG-Exp on three interactive benchmarks for LLM agents: ALFWorld (Shridhar et al., 2021), which consists of text-based household tasks requiring agents to navigate environments and manipulate objects through textual commands; WebShop (Yao et al., 2022a), which evaluates agents on online shopping tasks involving product search and purchase decisions under user-specified requirements; and ScienceWorld (Wang et al., 2022), which contains interactive science tasks that require agents to navigate rooms, manipulate objects, and conduct experiments based on scientific commonsense.

For each benchmark, we construct training and validation sets containing 100 and 30 instances, respectively. For ALFWorld, these instances are sampled according to the original data distribution, while evaluation is conducted on the original Seen and Unseen test sets. For WebShop and ScienceWorld, sampling from the original data distribution would leave many task categories underrepresented or entirely absent. Following the category-based evaluation settings in Gao et al. (2025b); Yang et al. (2025), we select the 20 largest task categories in WebShop and the 10 largest in ScienceWorld. For each benchmark, we sample the training and validation instances, along with 100 test instances, from the selected categories.

B.2 Baseline Summaries

Baselines. We consider five representative baselines:

- **No Experience.** The agent solves each task independently without using external experience from previous tasks. It therefore reflects the performance of the backbone LLM without experience augmentation.
- **ReasoningBank** (Ouyang et al., 2025). ReasoningBank distills past trajectories into structured memory items that capture useful reasoning strategies and lessons from failures. For

each new task, it retrieves the top-k most relevant memories using embedding similarity and adds them to the agent's system instruction. After the task is completed, an LLM-as-a-judge estimates whether the trajectory succeeds or fails, and the trajectory is then used to extract new memory items for future tasks.

- **MemP** (Fang et al., 2025). MemP extracts procedural memory from the trajectories and rewards of previously completed tasks. For a new task, it retrieves the memory associated with the most similar previous task using cosine similarity over task embeddings and provides it to the agent during execution. The memory library is maintained according to task feedback through addition, deletion, and modification of memory entries.

- **MemRL** (Zhang et al., 2026). MemRL maintains an Intent-Experience-Utility memory, where each entry stores past task experience, including the complete solution trace or execution trajectory, together with a learned utility score. For a new task, semantically similar memories are first retrieved by embedding similarity and then reranked using both similarity and utility. After execution, the utilities of the retrieved memories are updated according to environmental rewards, and new experience entries are constructed from the completed trajectories and added to the memory bank.

- **DeltaMem** (Tan et al., 2026a). DeltaMem maintains two separate memory structures: a Task-Tree for procedural skills and an Env-Tree for environment knowledge. Each episode is distilled into structured experience, and when a similar memory already exists, only the residual difference is stored instead of a complete trajectory. During retrieval, nodes are matched by embedding similarity with an additional penalty for failed experiences, and the root-to-node path is reconstructed as the context for the agent. Frequently successful residual paths are further consolidated into new root memories.

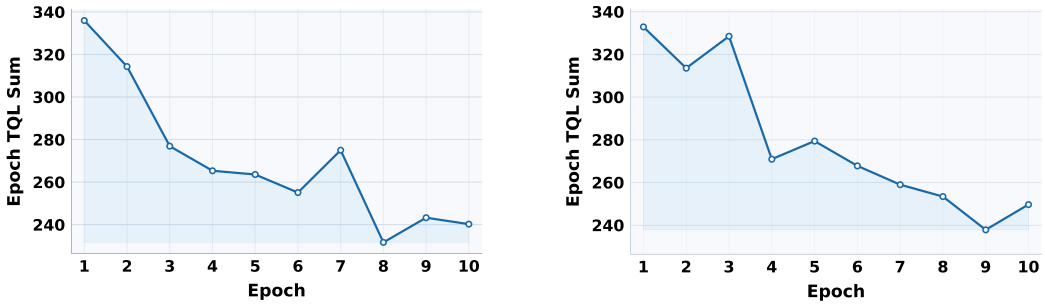
For all baselines, we follow the hyperparameter settings reported in their original papers. Our method uses eight rollouts per task, evenly divided between the baseline and experience-guided groups. We set $\alpha = 0.7$ and $c = 0.05$, retrieving the top-3 task-level and top-1 task-type-level experiences. For trajectory retrieval, we use the top-1 trajectory on WebShop and the top-2 on the other datasets. All methods are trained for 10 epochs, with a maximum of 30 interaction steps per task across the three datasets.

B.3 Performance and Interaction Efficiency across benchmarks

Table 4: Task Performance and Interaction Efficiency across Benchmarks using DeepSeek-V4-Flash.

Split	Method	Average@3 ↑	Avg. Steps ↓	Token Efficiency	
				Tokens / Episode ↓	Tokens / Success ↓
Webshop	No Experience	42.33	21.59	**47.20k**	**111.49k**
	MemRL	<u>73.67</u>	<u>12.60</u>	205.00k	278.28k
	TQG-Exp	**79.67**	**8.77**	<u>55.90k</u>	<u>70.16k</u>
Scienceworld	No Experience	47.00	18.77	**30.46k**	**64.80k**
	MemRL	<u>69.33</u>	<u>14.41</u>	220.61k	318.18k
	TQG-Exp	**74.67**	**12.46**	<u>60.82k</u>	<u>81.45k</u>

We further compare the task performance and interaction efficiency of TQG-Exp with MemRL on WebShop and ScienceWorld, as shown in Tab. 4. On WebShop, TQG-Exp improves Average@3 from 73.67 to 79.67 while reducing the average episode length from 12.60 to 8.77. Meanwhile, Tokens/Episode and Tokens/Success decrease by 72.7% and 74.8%, respectively. Similar improvements are observed on ScienceWorld, where TQG-Exp increases Average@3 from 69.33 to 74.67 and reduces the average episode length from 14.41 to 12.46. Tokens/Episode and Tokens/Success are further reduced by 72.4% and 74.4%, respectively. These results demonstrate that TQG-Exp achieves higher task performance than MemRL while requiring fewer interaction steps and substantially fewer tokens across both benchmarks.



(a) WebShop (b) ScienceWorld

Figure 6: Evolution of the epoch-level TQL sum on WebShop and ScienceWorld during training with DeepSeek-V4-Flash.

B.4 TQL-Sum Evolution on Additional Benchmarks

Fig. 6 shows the evolution of the epoch-level TQL sum of the experience-guided groups during training on WebShop and ScienceWorld with DeepSeek-V4-Flash. Across both benchmarks, the TQL sum follows an overall downward trend, with moderate fluctuations across epochs. This behavior resembles the optimization of a training objective: although the signal does not decrease monotonically, its overall reduction indicates that the utility updates increasingly favor experiences associated with higher-quality trajectories.

B.5 Experience-Bank Evolution

Fig. 7 shows the evolution of the task-level and task-type-level experience bank sizes during training on ALFWorld with DeepSeek-V4-Flash. The task-level bank increases from 45 experiences at Epoch 1 to its capacity of 300 at Epoch 7, after which its size remains fixed. In contrast, the type-level bank grows more gradually, from 17 experiences at Epoch 1 to 212 experiences at Epoch 10, without reaching the task-level capacity. Once the task-level experience bank reaches capacity, newly generated experiences are evaluated alongside existing ones using the UCB-based score $u(e) + c\sqrt{\frac{\log(1+N)}{1+n(e)}}$. Experiences with the lowest scores are removed first. Consequently, from Epoch 7 onward, the task-level bank maintains a nearly constant size of 300, while lower-ranked experiences are gradually replaced by newly generated ones.

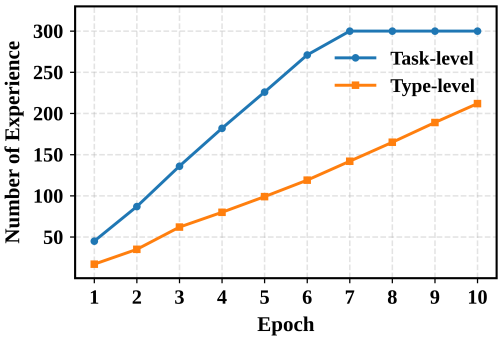


Figure 7: Sizes of the task-level and task-type-level experience banks on ALFWorld using DeepSeek-V4-Flash.

C Experience Extraction Prompts and Examples

C.1 Task-Level Experience Extraction Prompt

Failure-Aware Reflection

Role.
You are summarizing ordered stages for one task-level experience from representative trajectories.

Task Information.

```
Task: {item.get("task", "")}
Task Type: {item.get("task_type", "")}
Environment/Dataset: {env_name}
```

```
Missing representative types: {json.dumps(missing,
ensure_ascii=False)}
```

Representative Trajectories.

```
{self._format_optional_representative(
"BEST_SUCCESS_TRAJECTORY", item.get("tau_best_success"))}
{self._format_optional_representative(
"NEAR_MISS_FAILURE_TRAJECTORY", item.get("tau_near_failure"))}
{self._format_optional_representative(
"SEVERE_FAILURE_TRAJECTORY", item.get("tau_bad_failure"))}
```

Reflection Instruction.
Failure representatives are selected from baseline trajectories without retrieved experience, so use them to diagnose the base model's task mistakes.
Produce 1 to 3 ordered TASK_STAGE items as a JSON array.
Each item must describe one distinct, reusable stage-level lesson for this concrete task. Prefer causal mistakes over later symptoms: summarize a later invalid, repeated, or final wrong action only when it reveals an earlier missing check, wrong decision, or incorrect task understanding. Order items by when the lesson should be applied during task execution.

Required JSON Fields.

- `"order"`: integer, starting from 1
- `"stage"`: short environment-appropriate stage name
- `"title"`: short name of the experience
- `"principle"`: the concrete lesson
- `"when_to_apply"`: when this lesson should be used

Optional Fields.

- `"evidence"`: `"best_success"` | `"near_miss"` | `"severe_failure"` | `"mixed"`
- `"failure_to_avoid"`: common mistake this lesson prevents

Rules.

- Summarize only stage-level lessons for this concrete task.
- Do not force any fixed stage schema. Use stage names appropriate to the environment and task.
- Preserve execution order when the trajectory shows a clear order.
- Identify the earliest critical mistake or divergence that made later failure likely, and summarize the correction at the appropriate task stage.
- Use successful trajectories to identify effective key steps.
- Use near-miss failures to identify missing critical actions.
- Use severe failures to identify error patterns to avoid.
- The experience should be reusable for the same concrete task, not a verbatim copy of actions.
- Return only a valid JSON array and no extra text.

Failure-Only Reflection

Role.
You are summarizing ordered avoidance lessons for one task-level experience from failed trajectories only.

Task Information.

```
Task: {item.get("task", "")}
Task Type: {item.get("task_type", "")}
Environment/Dataset: {env_name}
Selection case: failure_only
```

```
Missing representative types: {json.dumps(missing,
ensure_ascii=False)}
```

Representative Trajectories.

```
{self._format_optional_representative(
"LOWEST_TQL_FAILURE_TRAJECTORY", item.get("tau_near_failure"))}
{self._format_optional_representative(
"HIGHEST_TQL_FAILURE_TRAJECTORY", item.get("tau_bad_failure"))}
```

Reflection Instruction.
The failed trajectories are selected from baseline attempts without retrieved experience. Treat them as evidence of base-model task mistakes.
Compare the failed trajectories for the same task.
The lowest-TQL failure is the least bad failed attempt and may show partial progress.
The highest-TQL failure is the worst failed attempt and may show severe error patterns.
Extract reusable task-level experience about what to avoid and how to correct the next attempt.
Produce 1 to 3 ordered TASK_STAGE items as a JSON array.
Prefer early task stages where the failed trajectories begin to drift toward failure.
These items will be stored together as stages inside one task-level experience.
Each item must describe one distinct, reusable avoidance or correction lesson for this concrete task.
Prefer earlier causal mistakes when the evidence supports them, but do not force an early cause if the failure evidence is only visible later.
The items must be ordered by when the correction should be applied during task execution.

Required JSON Fields.

- `"order"`: integer, starting from 1
- `"stage"`: short environment-appropriate stage name
- `"title"`: short name of the avoidance lesson
- `"principle"`: the concrete corrective lesson
- `"when_to_apply"`: when this lesson should be used

Optional Fields.

- `"evidence"`: `"lowest_tql_failure"` | `"highest_tql_failure"` | `"mixed"`
- `"failure_to_avoid"`: common mistake this lesson prevents

Rules.

- Summarize only stage-level lessons for this concrete task.
- The trajectories shown are failures; do not describe their actions as successful.
- Do not invent any successful outcome, unseen state, hidden observation, or successful action sequence that is not supported by the provided failed trajectories.
- Focus on avoidable mistakes such as misunderstanding the task goal, choosing an invalid or unhelpful action, using poor search/query/filter choices when applicable, ignoring key constraints or observations, repeating actions, acting in the wrong order, stopping too early, and missing necessary verification.
- When possible, identify an early mistake or recurring failure pattern that made later failure more likely, and summarize the correction at the appropriate task stage.
- Prefer concrete correction rules that would make the next attempt less likely to repeat the observed failures.
- The experience should be reusable for the same concrete task, not a verbatim copy of failed actions.
- If the failed trajectories do not support any reliable lesson, return [].
- Return only a valid JSON array and no extra text.

Success-Contrast Reflection

Role.
You are summarizing ordered stages for one task-level experience by comparing two successful trajectories.

Task Information.

```
Task: {item.get("task", "")}
Task Type: {item.get("task_type", "")}
Environment/Dataset: {env_name}
Selection case: success_contrast
Missing representative types: {json.dumps(missing,
ensure_ascii=False)}
```

Representative Trajectories.

```
{self._format_optional_representative(
"BEST_SUCCESS_TRAJECTORY", item.get("tau_best_success"))}
{self._format_optional_representative(
"WORST_SUCCESS_TRAJECTORY", item.get("tau_worst_success"))}
```

Reflection Instruction.
The worst successful trajectory is selected from baseline successful attempts, so focus on improving the base model's successful trajectory quality.
Compare two successful trajectories for the same task.
The best success has lower TQL and represents a cleaner solution.
The worst success also succeeded but has higher TQL, likely due to extra steps, repeated actions, invalid actions, or inefficient ordering.
Extract reusable task-level experience that preserves the successful strategy while avoiding inefficient successful patterns.
Produce 1 to 3 ordered TASK_STAGE items as a JSON array. These items will be stored together as stages inside one task-level experience.
Each item must describe one distinct, reusable stage-level lesson for this concrete task.
The items must be ordered by the stage/order in which the lesson applies during task execution.

Required JSON Fields.

- `"order"`: integer, starting from 1
- `"stage"`: short environment-appropriate stage name
- `"title"`: short name of the experience
- `"principle"`: the concrete lesson
- `"when_to_apply"`: when this lesson should be used

Optional Fields.

- `"evidence"`: `"best_success"` | `"worst_success"` | `"mixed"`
- `"failure_to_avoid"`: inefficient successful pattern this lesson prevents

Rules.

- Summarize only stage-level lessons for this concrete task.
- Do not force any fixed stage schema. Use stage names appropriate to the environment and task.
- Preserve execution order when the trajectory shows a clear order.
- Do not describe WORST_SUCCESS_TRAJECTORY as a failure.
- Do not invent failure modes.
- Focus on efficiency, necessary ordering, redundant actions, invalid actions, repeated actions, and stage-level improvements.
- The experience should be reusable for the same concrete task, not a verbatim copy of actions.
- Return only a valid JSON array and no extra text.

C.2 TASK-TYPE EXPERIENCE EXTRACTION PROMPT

Task-Type-Level Experience Distillation

Role.
You are abstracting task-type-level stage experiences from staged task-level experiences.

Task Information.

```
Task Type: {task_type}
Environment/Dataset: {env_name}
```

Buffered Task-Level Experiences.

```
Buffered staged task-level experiences:
{json.dumps(payload, ensure_ascii=False, indent=2)}
```

Distillation Instruction.
Each task-level experience describes one concrete task with task_tql, task_goal, and ordered stages.
Your job is to extract common reusable stage-level patterns from multiple tasks of the same task_type.
Produce 1 to 3 task-type-level stage experiences as a JSON array.
Each item must correspond to one distinct reusable stage, decision point, or failure-avoidance pattern.
Do not create duplicate experiences.
Do not create multiple items for the same target step/stage.
Use distinct `"common_order_or_stage"` values across all items.
If two candidate items would apply to the same stage, merge them into one stronger item.

Required JSON Fields.

- `"title"`
- `"principle"`
- `"when_to_apply"`
- `"common_order_or_stage"`
- `"critical_steps"`
- `"common_failure_patterns"`
- `"avoidance_rules"`

Rules.

- Extract only from the provided staged task-level experiences.
- Do not summarize individual tasks separately.
- Do not infer directly from raw trajectories except through the provided task-level experiences.
- Merge semantically similar stages even if objects, products, or environment details differ.
- `"common_order_or_stage"` must be a concise semantic stage label, not a bare number or generic label such as `"1"`, `"2"`, `"stage 1"`, or `"step 2"`.
- Prefer patterns supported by multiple task experiences.
- Include a highly important single-source pattern only if it is central to the task_type.
- Return only a valid JSON array and no extra text.

C.3 EXPERIENCE EXAMPLES

Example of Task-Level Experience on ALFWorld with DeepSeek-V4-Flash

```
{
  "title": "Task procedure for heat some mug and put it in
coffeemachine.",
  "principle": "Prioritize checking visible surfaces like countertops
for the mug, as it is often placed there. Avoid exhaustive cabinet
and drawer searches unless necessary. After obtaining the mug, go to
```

```
the microwave, open it, place the mug inside, close the microwave,
then execute the heat action.  Omitting the heat step after closing
the microwave is a critical error.  After heating, take the heated
mug from the microwave, go to the coffeemachine, and put the mug in
it.  The task requires both heating and placement; omitting the final
placement leaves the task incomplete.",
  "when_to_apply":  "When solving this concrete task or a very similar
task:  heat some mug and put it in coffeemachine.",
  "experience_level":  "task",
  "task_type":  "pick_heat_then_place_in_recep",
  "source_task":  "heat some mug and put it in coffeemachine.",
  "retrieval_obs":  "heat some mug and put it in coffeemachine.",
  "stages":  [
    {
      "order":  1,
      "stage":  "Object Acquisition",
      "title":  "Locate and take mug from countertop",
      "principle":  "Prioritize checking visible surfaces like
countertops for the mug, as it is often placed there.  Avoid
exhaustive cabinet and drawer searches unless necessary.",
      "when_to_apply":  "At the start of the task, after initial
observation.",
      "evidence":  "mixed",
      "failure_to_avoid":  "Unnecessary searching of multiple cabinets
and drawers when the mug is already visible on a countertop."
    },
    {
      "order":  2,
      "stage":  "Heating",
      "title":  "Heat mug in microwave after placing it inside",
      "principle":  "After obtaining the mug, go to the microwave,
open it, place the mug inside, close the microwave, then execute the
heat action.  Omitting the heat step after closing the microwave is a
critical error.",
      "when_to_apply":  "After taking the mug, before moving to the
coffeemachine.",
      "evidence":  "near_miss",
      "failure_to_avoid":  "Placing mug in microwave and closing it
without performing the explicit 'heat' action."
    },
    {
      "order":  3,
      "stage":  "Placement",
      "title":  "Place heated mug in coffeemachine",
      "principle":  "After heating, take the heated mug from the
microwave, go to the coffeemachine, and put the mug in it.  The task
requires both heating and placement; omitting the final placement
leaves the task incomplete.",
      "when_to_apply":  "After successfully heating the mug.",
      "evidence":  "mixed",
      "failure_to_avoid":  "Stopping after heating or becoming stuck in
repetitive actions without proceeding to place the mug in the
coffeemachine."
    }
  ],
```

```
  "num_stages": 3,
  "stage_when_to_apply": [
    "At the start of the task, after initial observation.",
    "After taking the mug, before moving to the coffeemachine.",
    "After successfully heating the mug."
  ],
  "failure_to_avoid": "Unnecessary searching of multiple cabinets and
drawers when the mug is already visible on a countertop.; Placing mug
in microwave and closing it without performing the explicit 'heat'
action.; Stopping after heating or becoming stuck in repetitive
actions without proceeding to place the mug in the coffeemachine.",
  "env_name": "alfworld_parameter_free",
  "data_source": "alfworld_parameter_free",
  "utility": 0.4609896854962928,
  "retrieval_count": 37,
  "last_retrieval_step": 66,
  "created_at_step": 28,
  "skill_id": "task_183"
}
```


Example of Task-Type-Level Experience on ALFWorld with DeepSeek-V4-Flash


```
{
  "title": "Place Heated Object in Target Receptacle",
  "principle": "After the object is successfully heated, retrieve it
from the microwave if not already in inventory. Navigate directly to
the target receptacle, ensuring it is open if necessary. Use a
move/put action (e.g., 'move <object> to <receptacle>') to place the
object. Immediately verify that the object is no longer in inventory
and appears at the receptacle. If placement failed (object still in
inventory), retry the move action. Do not explore other objects or
areas until placement is confirmed.",
  "when_to_apply": "Immediately after the heating action succeeds and
the object is hot, before performing any other non-essential
actions.",
  "common_order_or_stage": "Final Placement",
  "critical_steps": [
    "If the object is still inside the microwave, open the microwave
and take it out to have it in inventory",
    "Navigate to the target receptacle",
    "Check if the receptacle is open; if not, open it",
    "Verify that the object in inventory is hot (e.g., by recent heat
action or inventory state)",
    "Use 'move [object] to [receptacle]' or 'put [object] on
[receptacle]' to place the object",
    "Check inventory or observe the receptacle to confirm placement
succeeded",
    "If the object remains in inventory, retry the move action"
  ],
  "common_failure_patterns": [
    "Stopping after heating without placing the object",
    "Assuming placement succeeded without verification",
    "Performing unnecessary actions (examining other objects, opening
containers) after heating",
```

```
    "Getting stuck in loops of opening/closing the microwave or other
containers",
    "Incorrectly identifying the target receptacle (e.g., using wrong
name)",
    "Forgetting to open the receptacle before moving the object",
    "Placing a wrong object (e.g., cup instead of mug)",
    "Placing the object before heating (unheated placement)",
    "Remaining at the microwave or wandering to other containers after
heating instead of proceeding to the target receptacle"
  ],
  "avoidance_rules": [
    "After heating, immediately retrieve the object if not already in
inventory",
    "Navigate directly to the target receptacle without detours",
    "Always ensure the receptacle is open before placing (open it if
necessary)",
    "Before placing, confirm that the object is hot and matches the
task's exact name",
    "Verify placement by checking inventory (should be empty of the
object) or observing the receptacle's contents",
    "If the object remains in inventory, retry the move action",
    "Do not explore other areas or objects until placement is
confirmed"
  ],
  "experience_level": "type",
  "task_type": "pick_heat_then_place_in_recep",
  "retrieval_obs": "Task type: pick_heat_then_place_in_recep\nStage:
Final Placement\nWhen: Immediately after the heating action succeeds
and the object is hot, before performing any other non-essential
actions.\nGoal: Place Heated Object in Target Receptacle",
  "last_operation": "merge",
  "skill_id": "step_020",
  "utility": 0.75,
  "retrieval_count": 2,
  "last_retrieval_step": 18,
  "created_at_step": 7,
  "last_modified_step": 28,
  "version": 5,
  "merged_count": 4
}
```