LEARNING TO DISTILL EXPERIENCE: UTILITY-DRIVEN EXPERIENCE INDUCTION FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly distill interaction trajectories into external memory to reuse past experience, yet determining what should actually be learned remains challenging. A single trajectory can support multiple plausible lessons, while its outcome provides limited evidence about which lessons will help in subsequent decision contexts. We formulate this challenge as *utility-driven memory induction*: learning to transform interaction trajectories into reusable knowledge by optimizing its downstream decision utility, rather than trajectory-level outcomes alone.. We propose **EvoExp**, a trainable experience inducer that generates actionable guidance paired with explicit applicability conditions. Following supervised initialization, **Experience-Utility GRPO (EU-GRPO)** optimizes the inducer through paired evaluations of a frozen actor under current and updated memory stores, using estimated action-value improvements across decision contexts as feedback. Once trained, EvoExp constructs new memories from fresh trajectories without further parameter updates or a deployment-time value evaluator. Across five strategic games, the RL-trained inducer outperforms the no-memory control in every game and achieves the best reported performance in three, including a **+0.20** paired win-rate gain on Dou Dizhu and **+10.30** chips on Texas Hold'em; despite RL training only on Dou Dizhu, EvoExp further improves on ALFWorld and GPT-authored Tide Ledger game.

1 INTRODUCTION

Large language model (LLM) agents increasingly use external memory to preserve knowledge from interaction and inform later decisions (Ouyang et al., 2026; Yan et al., 2026; Yu et al., 2026; Cai et al., 2026). The usefulness of this memory depends on how interaction trajectories are transformed into guidance: an agent must identify what an experience teaches and determine how that lesson should influence subsequent behavior. This makes experience induction a learning problem in its own right. We study how to train an experience-to-memory transformation that, once learned, constructs useful memories from new interaction histories to improve a fixed actor's subsequent decisions.

As shown in Figure 1, a trajectory does not uniquely determine what should be learned. The same successful action may reflect a transferable strategy, a particular opponent or state configuration, or an incidental event. Thus, several lessons may explain the observed outcome while leading to different decisions when reused elsewhere. A broadly stated recommendation may help in one context and mislead the actor in another, whereas an overly specific lesson may rarely apply again. Conversely, an unsuccessful trajectory may still contain useful decision knowledge. The source outcome thus provides limited evidence about what to retain, how broadly to abstract it, and where the resulting guidance should be used.

Prior work has demonstrated the value of both distilling experience (Wang et al., 2024; Zhao et al., 2024) and learning memory updates (Zhang et al., 2026a; Yan et al., 2026). ReasoningBank (Ouyang et al., 2026) extracts reusable reasoning strategies from self-evaluated successes and failures, while MemoPilot (Cai et al., 2026) trains a memory updater to improve a frozen player using rewards from subsequent games. These approaches motivate a more specific question about the feedback used to train memory induction. Even when feedback comes from subsequent use, an episode-level outcome combines multiple decisions and environment dynamics, making the incremental effect of a memory update difficult to assess. We therefore assign credit to candidate memory updates through

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

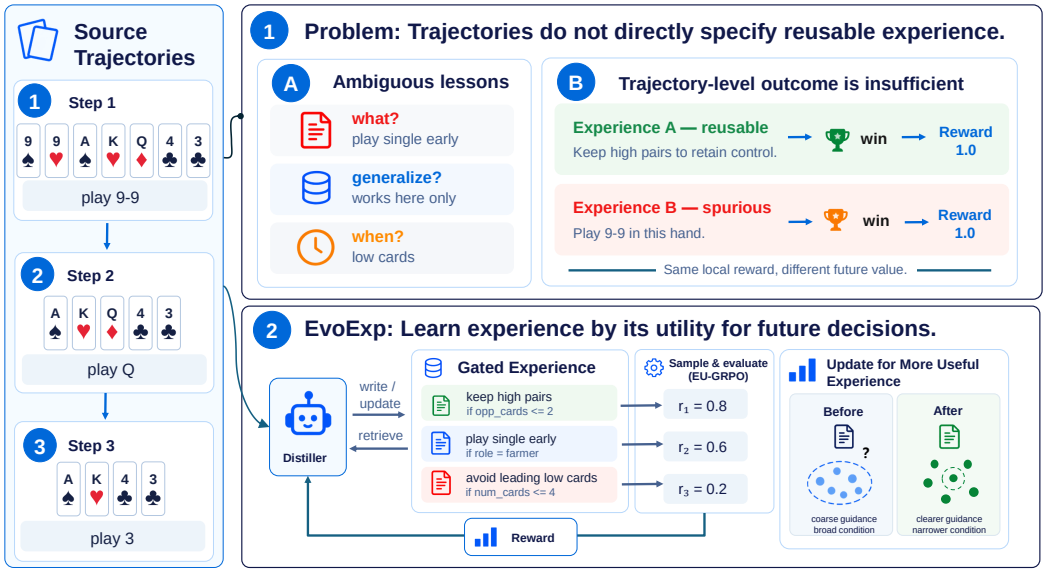


Figure 1: Motivation for utility-driven memory induction. A trajectory can support multiple plausible lessons, while its outcome alone provides limited evidence about which lessons will help in future decision contexts. EvoExp addresses this ambiguity by learning to construct guidance paired with applicability conditions, using the downstream decision utility of candidate memory updates as training feedback.

their effects on decisions in sampled reuse contexts, comparing a fixed actor under the current and updated stores at the same states.

We formulate this problem as *utility-driven memory induction*, in which an inducer learns to transform interaction trajectories into reusable knowledge by optimizing the downstream decision utility of the resulting memory. Under this formulation, what to extract, how to abstract it, and when to expose it are coupled induction choices. We propose **EvoExp**, a trainable experience inducer that maps interaction trajectories and the current memory store to structured add, revise, and remove operations. Its strategic guidance is paired with executable applicability conditions that specify the observable contexts in which it is made available to the actor. The guidance and its condition together determine how an update changes the actor's decision context, allowing both memory content and selective exposure to be optimized through the same objective.

Following supervised initialization, we optimize EvoExp with *Experience-Utility GRPO (EU-GRPO)*, which combines standard GRPO with paired decision-utility rewards. For each candidate update, we evaluate the same frozen actor under the current and updated memory stores at a common collection of intermediate decision states. A fixed action-value evaluator scores the selected actions, and their estimated value differences are aggregated into a reward for the update. This evaluates memory through its effects on decisions across the sampled contexts and allows both beneficial and harmful uses of guidance to influence training. Training updates the inducer while keeping the actor and evaluator fixed. Once trained, EvoExp constructs new memories from fresh trajectories without further parameter updates or a value evaluator for memory induction in the deployment environment.

We evaluate EvoExp with a frozen Qwen3-8B actor across five strategic games. The RL-trained inducer improves over the no-memory control in every game and achieves the highest performance among compared methods in three. It also improves over supervised initialization in four of the five games. Although RL training uses only Dou Dizhu, evaluations on Tide and ALFWorld, both excluded from inducer training, show additional gains over the supervised inducer. These results suggest that improvements to experience induction can transfer beyond its training environments. Reward-design comparisons and applicability studies further examine how decision-utility feedback and selective memory exposure contribute to downstream performance.

Our main contributions are summarized as follows:

- We formulate *utility-driven memory induction*, treating the transformation from interaction trajectories to reusable knowledge as a learning target evaluated through downstream decision utility.
- We introduce **EvoExp** and **EU-GRPO**, which optimize memory updates containing actionable guidance and applicability conditions through paired evaluations of a frozen actor across decision states.
- We conduct extensive experiments showing improved downstream performance and transfer of the learned induction procedure to environments excluded from training, complemented by analyses of reward design and memory applicability.

2 Related Work

External memory and reusable experience. External-memory systems preserve information beyond an agent's current context and reintroduce it during later decisions. Generative Agents stores observations and higher-level reflections (Park et al., 2023); MemoryBank (Zhong et al., 2024) maintains long-term conversational memories with reinforcement and forgetting; MemGPT (Packer et al., 2023) manages tiered context through model-generated operations; and A-MEM (Xu et al., 2026a) creates linked notes whose attributes and relations evolve over time. Other systems convert trajectories into more explicitly reusable artifacts. Reflexion (Shinn et al., 2023) writes verbal feedback after a trial, ExpeL (Zhao et al., 2024) aggregates successful and failed experience into transferable textual insights, Voyager (Wang et al., 2023) builds a library of verified executable skills, and ReasoningBank (Ouyang et al., 2026) distills generalizable reasoning strategies. These works demonstrate that experience stored in external memory can improve an agent's decisions without updating its action model. We build on this line of work by learning to transform trajectories into reusable guidance, jointly optimizing its content and applicability conditions for downstream decision utility.

Learning memory construction and use. Recent methods learn or adapt memory construction, management, and retrieval. Memory-R1 (Yan et al., 2026) learns structured memory operations using downstream outcome rewards, while AgeMem (Yu et al., 2026) incorporates short- and long-term memory operations into the agent's action space and optimizes their joint management. MemSkill (Zhang et al., 2026a) trains a controller to select memory-construction skills and evolves the skill set through an LLM designer. MemRL (Zhang et al., 2026b) updates utility estimates for episodic memories to guide retrieval with a frozen LLM. These methods address distinct learning targets, including memory operations, construction skills, and retrieval utility. Our proposed EvoExp learns to generate structured store updates from trajectories, pairing guidance with executable applicability conditions that control its exposure to a fixed actor.

Credit assignment for memory utility. Recent work develops finer-grained supervision for memory utility. MemBuilder (Shen et al., 2026) introduces session-level questions and attributes reward across memory components, while Mem-T (Yue et al., 2026) propagates evidence from later retrieval and answer paths to earlier construction operations. MemPO (Li et al., 2026) provides memory-specific supervision based on the likelihood of correct downstream answers. In interactive games, MemoPilot (Cai et al., 2026) trains a plug-in updater for a frozen actor using subsequent-game rewards and multi-turn policy optimization. MemGym (Xu et al., 2026b) evaluates memory interventions under fixed reasoners and trains an evaluator to predict whether context compression preserves behavior. Our proposed EvoExp focuses on evaluating candidate store updates across multiple reuse contexts. It compares a frozen actor's selected actions under the current and updated stores at shared intermediate decision states, aggregating their estimated action-value differences into an update-level reward. This objective trains the inducer through the combined effect of generated guidance and its executable exposure conditions.

3 Methodology

EvoExp pairs a frozen actor with a trainable experience inducer that maps interaction trajectories and existing memory to structured store updates. These updates contain actionable guidance and executable applicability conditions. Following supervised initialization, Experience-Utility GRPO

(EU-GRPO) optimizes the inducer using estimated action-value differences of the actor under current and updated memory stores across sampled decision states.

3.1 Problem Formulation: Experience Induction

Let π_ϕ be a fixed actor that receives observation o_t and selects an action a_t from the legal action set $\mathcal{A}_t$, producing a trajectory

$$\tau = (o_1, \mathcal{A}_1, a_1, \ldots, o_T, \mathcal{A}_T, a_T). \tag{1}$$

EvoExp transforms accumulated trajectories into an external experience store $\mathcal{M}$. Each entry is represented as

$$e_i = (\eta_i, c_i, g_i, r_i) \in \mathcal{M}, \tag{2}$$

where η_i is metadata, c_i is a machine-evaluable applicability condition, g_i is actionable natural-language guidance, and r_i is its rationale.

We interpret (g_i, c_i) as a conditional decision intervention. The guidance g_i specifies a behavioral intervention, while c_i specifies the validity region in which that intervention is allowed to enter the actor's context. Applicability is therefore part of the meaning of an induced experience, not a separate retrieval enhancement.

Given accumulated interaction evidence $\mathcal{T}_k$ and the current store $\mathcal{M}_k$, the inducer samples an update

$$u_k \sim q_\theta(\cdot \mid \mathcal{T}_k, \mathcal{M}_k), \qquad \widetilde{\mathcal{M}}_k = \mathcal{U}(\mathcal{M}_k, u_k), \tag{3}$$

where u_k may add, replace, or remove entries and $\mathcal{U}$ is the deterministic update interface. The objective is to learn updates that improve the frozen policy's decisions in future states, rather than to reconstruct or summarize the source trajectory.

3.2 EvoExp Framework

Given interaction evidence and an experience store, EvoExp realizes experience induction through experience construction, applicability-aware retrieval, and experience-guided decision making.

Experience Construction. At each update, EvoExp provides the accumulated trajectories and current experience store to the inducer, which outputs structured store operations. Each induced entry pairs guidance g_i, which abstracts decision-relevant knowledge, with an executable condition c_i that determines when the guidance is exposed. The inducer maintains the experience pool through tool calls that add new entries, revise existing ones, or remove obsolete ones, rather than simply appending a new summary after every trajectory.

Applicability-Aware Retrieval. At decision step t, a deterministic environment adapter maps the observable context into a structured representation,

$$z_t = f_{\text{env}}(o_t, \mathcal{A}_t, \rho_t^{\text{obs}}), \tag{4}$$

where ρ_t^{obs} is the observable interaction history. Each condition c_i is a restricted Python Boolean expression over z_t. The adapter exposes only information observable to the agent; available fields and validation rules are detailed in Appendix A.2. EvoExp retrieves conditional entries whose conditions hold,

$$\mathcal{R}_t^{\text{cond}} = \{e_i \in \mathcal{M}_{\text{cond}} \mid c_i(z_t) = 1\}. \tag{5}$$

Experience-Guided Decision Making. Let $P(h_t, \mathcal{M})$ denote the decision prompt formed from the observable context $h_t = (o_t, \mathcal{A}_t, \rho_t^{\text{obs}})$ and retrieved memory. For conditional entries, the prompt includes their guidance and rationale; applicability conditions serve only as retrieval gates. The frozen actor selects an action according to

$$a_t \sim \pi_\phi(\cdot \mid P(h_t, \mathcal{M})). \tag{6}$$

Memory thus influences decisions through the actor's input context, while the actor retains control over action selection.

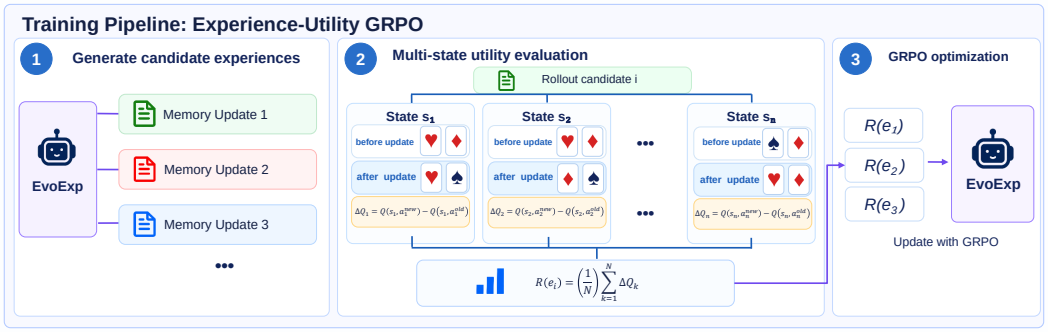


Figure 2: Overview of EU-GRPO. Candidate updates are evaluated by comparing a frozen actor under current and updated memory stores at matched decision states. Estimated action-value differences provide rewards for training the inducer.

3.3 Learning the Experience Inducer

The experience-induction formulation defines a conditional knowledge construction task: given interaction evidence and an existing experience store, predict update operations that extract reusable guidance together with the conditions under which that guidance should apply.

Supervised Experience Construction. To train the inducer without manually annotated updates, we use Qwen3.8-Flash (Qwen Team, 2026) to run game trajectories through the same experience-construction pipeline. We collect the prompts and corresponding store-update responses as SFT examples. We retain 5,000 final candidate records covering Blackjack, Dou Dizhu, Liar's Dice, Love Letter, and Texas Hold'em; Big Two is not included. We fine-tune EvoExp for one epoch using the standard token-level autoregressive cross-entropy loss on the target update sequences. This SFT stage initializes experience construction, after which reinforcement learning optimizes the inducer for downstream decision utility.

3.4 Credit Assignment at the Point of Reuse

Following supervised initialization, we optimize the inducer with *Experience-Utility GRPO (EU-GRPO)*. As shown in Figure 2, we evaluate candidate memory updates by comparing a frozen actor under current and updated stores at matched intermediate decision states. The resulting estimated action-value differences provide update-level rewards for learning the trajectory-to-memory transformation within standard GRPO (Shao et al., 2024).

Intermediate Decision States. For each reinforcement-learning step, we sample a collection of intermediate decision states,

$$\mathcal{H} = \{h_s\}_{s=1}^{N}. \tag{7}$$

Each h_s contains the information required to reproduce the actor's decision, including the current observation, legal actions, and observable interaction history. We additionally assume access to a fixed environment-specific action-value evaluator

$$\widehat{Q}_{\text{env}}(h_s, a), \tag{8}$$

which estimates the downstream utility of taking legal action a at state h_s.

The action-value evaluator is environment dependent and is separate from both the experience inducer and the decision policy. Its role is to measure the downstream utility of the actor's decisions under different experience contexts: it neither generates experience nor selects actions for the decision model. For environments with a tractable transition model, action values can be computed exactly or approximately through dynamic programming. For more complex environments, they may be provided by a pretrained value function. Importantly, these values are used only to evaluate the actions produced by the frozen actor and are never provided to the actor itself.

Candidate Experience Updates. Let M_0 be the experience store before the current update and $x = (T, M_0)$ the training context. Following GRPO, we roll out a group of candidate store operations

$$\{u_i\}_{i=1}^{G} \sim q_\theta(\cdot \mid x), \qquad M_i = \mathcal{U}(M_0, u_i). \tag{9}$$

Each u_i therefore defines a candidate store M_i. Invalid operations are discarded, while a well-formed no-op remains a valid candidate.

Paired Decision-Utility Feedback. For each intermediate state h_s, we evaluate the frozen policy with the baseline store M_0 and with the candidate store M_i. Specifically, for $j \in \{0, i\}$, we retrieve from M_j, construct the normal decision prompt $P_j(s)$, and sample

$$a_{j,s}^{(k)} \sim \pi_\phi(\cdot \mid P_j(s)), \qquad k = 1, \ldots, K. \tag{10}$$

The fixed environment evaluator scores these actions, and we average its feedback over the K samples:

$$\overline{Q}_j(s) = \frac{1}{K} \sum_{k=1}^{K} \widehat{Q}_{\text{env}}\left(h_s, a_{j,s}^{(k)}\right). \tag{11}$$

Thus, the paired evaluations hold the state, actor, evaluator, and decoding procedure fixed; they differ only in whether the candidate experience update is applied. We use their difference as the candidate's reward:

$$R_i = \frac{1}{N} \sum_{s=1}^{N} \mathbf{1}[P_i(s) \neq P_0(s)] \left(\overline{Q}_i(s) - \overline{Q}_0(s)\right). \tag{12}$$

If the candidate does not change the prompt, it receives zero credit at that state. Consequently, R_i measures whether exposing the candidate experience causes the frozen policy to select higher-value actions at its points of reuse.

GRPO Update. Let $\mathcal{V}$ be the set of valid candidates. We normalize their rewards within the rollout group,

$$A_i = \frac{R_i - \mu_{\mathcal{V}}}{\sigma_{\mathcal{V}} + \epsilon}, \qquad i \in \mathcal{V}, \tag{13}$$

and use the resulting advantages in the standard clipped GRPO objective:

$$\mathcal{J}_{\text{EU-GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \frac{1}{|u_i|} \sum_{t=1}^{|u_i|} \min\left(\rho_{i,t} A_i, \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon) A_i\right)\right], \tag{14}$$

where

$$\rho_{i,t} = \frac{q_\theta(u_{i,t} \mid x, u_{i,<t})}{q_{\theta_{\text{old}}}(u_{i,t} \mid x, u_{i,<t})}. \tag{15}$$

Backpropagation updates only the experience inducer q_θ; the decision policy π_ϕ and value evaluator $\widehat{Q}_{\text{env}}$ remain fixed. In our experiments, RL is performed only in Dou Dizhu, where $\widehat{Q}_{\text{env}}$ is a fixed DouZero (Zha et al., 2021) evaluator and its `ev_mean` output provides the feedback above. The trained inducer is then evaluated in the other environments without further RL optimization.

4 Experiments

4.1 Experimental Setup

Environments and evaluation protocol. We evaluate on five strategic game environments: Big Two, Dou Dizhu, Liar's Dice, Love Letter, and Texas Hold'em. Big Two is unseen during SFT, and EU-GRPO is trained only on Dou Dizhu. Across all evaluations, we use the same frozen Qwen3-8B actor, and all opponents are instantiated as their corresponding no-experience versions. For each game and method, we construct ten experience stores from disjoint 50-episode source groups. Each frozen store is evaluated on 400 unseen initial scenarios balanced across positions. Methods

share the ordered source episodes and test scenarios but interact with the environment independently during evaluation.

Our primary metric is the paired change in terminal return relative to the same actor without persistent experience. For a store M and frozen test scenarios $x_1, \ldots, x_E$, we report

$$\widehat{U}(M) = \frac{1}{E} \sum_{i=1}^{E} \left[G(M; x_i) - G(\emptyset; x_i) \right], \qquad E = 400. \tag{16}$$

For Texas Hold'em, G is terminal chip payoff. For Big Two, Dou Dizhu, Liar's Dice, and Love Letter, G is the terminal win indicator, so the reported quantity is a win-rate change.

Comparison groups. We separate game-specific *reference policies* from matched experience methods. Reference policies include RuleAgent, GPT-Generated Experience, and CFR/MCCFR (Zinkevich et al., 2007; Lanctot et al., 2009). Matched frozen-actor comparisons include Reflexion, ExpeL, MemoryBank, ReasoningBank, and EvoExp (Shinn et al., 2023; Zhao et al., 2024; Zhong et al., 2024; Ouyang et al., 2026). All LLM-based comparisons use the same frozen Qwen3-8B Yang et al. (2025) actor, retrieval procedure, and decoding configuration; only the method or model that constructs the experience store changes. We additionally compare EvoExp initialized from Qwen3-8B Base, its SFT checkpoint, and its RL checkpoint.

Implementation details. We train the EU-GRPO inducer from Qwen3-8B for 150 optimization steps, sampling 16 candidate updates per training context. All formal comparisons freeze the actor. Complete hyperparameters appear in Appendix A.1, and the action-generation and store-update prompts appear in Appendix A.3.

4.2 Overall Decision-Making Performance

Table 1 evaluates whether better experience induction improves downstream decisions. Off-the-shelf Qwen3.8-Flash and DeepSeek-V4-Flash constructors produce positive EvoExp gains in all five games, whereas prompt-based experience methods are less consistent. This indicates that pairing guidance with an enforced applicability boundary is a useful interface even before local training.

Training the inducer yields further numerical improvements. SFT exceeds the base inducer in four of five games. RL then trends upward relative to SFT on Dou Dizhu, Liar's Dice, Texas Hold'em, and Love Letter, while decreasing on Big Two. EvoExp-RL remains positive in every environment and is best overall in three of five. The table reports point estimates; across the experiments, the average confidence-interval half-width is ± 3.72 for Texas Hold'em and ± 0.07 for the other four games. Differences on the scale of the interval, including the Dou Dizhu SFT–RL gap, should therefore be interpreted as numerical trends rather than conclusive pairwise improvements.

4.3 Where Should Experience Receive Credit?

The central experimental question is not simply which reward is denser, but which event should provide credit to an induced lesson. We compare three signals under the same SFT initialization, training contexts, candidate updates, frozen actor, and GRPO settings:

- *Next-Game Payoff* assigns trajectory-level credit using the outcome of the subsequent complete game, following the reward unit used by MemoPilot (Cai et al., 2026).
- *State Rollout* begins from the same sampled states but propagates credit through subsequent actions and environment dynamics to a terminal return.
- *Decision Utility* assigns direct reuse-level credit using the paired change in action value caused by the candidate experience at those states.

As shown in Table 2, Decision Utility performs best in all three evaluated environments. State Rollout produces smaller gains, while Next-Game Payoff is weakest and becomes negative in two environments. Because the initialization, candidates, actor, and optimizer are matched, the comparison isolates the answer to "where should experience receive credit?" The results support assigning credit at the decisions where the lesson is reused, rather than propagating the outcome of the trajectory that follows its construction.

Table 1: Paired change in terminal return relative to the same actor without persistent experience. Texas Hold'em is measured in chips; the other games are measured in win-rate change.

Framework	Dou Dizhu	Liar's Dice	Texas Hold'em	Big Two	Love Letter
Reference Policies and Direct Experience					
RuleAgent	+0.14	+0.12	-0.53	+0.46	**+0.48**
CFR/MCCFR	–	+0.38	-0.34	+0.48	-0.02
Direct GPT Experience	+0.12	+0.46	+4.00	+0.36	+0.28
Qwen3.8-Flash As Experience Inducer					
ExpeL	-0.04	+0.38	-3.28	+0.09	+0.26
ReasoningBank	+0.17	+0.70	-7.37	-0.08	+0.31
Reflexion	+0.02	+0.72	-5.00	+0.08	+0.26
MemoryBank	-0.06	+0.27	-8.88	-0.01	+0.20
EvoExp	+0.02	+0.63	+6.36	+0.12	+0.25
DeepSeek-V4-Flash As Experience Inducer					
ExpeL	-0.06	+0.14	-2.34	+0.05	+0.20
ReasoningBank	+0.15	+0.38	+4.12	+0.54	+0.26
Reflexion	+0.00	+0.48	-2.54	+0.12	+0.18
MemoryBank	-0.04	+0.27	-6.23	+0.17	+0.17
EvoExp	+0.02	+0.36	+6.14	**+0.70**	+0.28
Locally Trained Inducers					
EvoExp + Qwen3-8B Base	+0.09	+0.48	-2.65	+0.10	+0.17
EvoExp + SFT	+0.14	+0.56	+8.05	+0.38	+0.14
EvoExp + RL	**+0.20**	**+0.76**	**+10.30**	+0.30	+0.31

Table 2: Matched comparison of credit-assignment units. Values are paired changes in terminal return relative to the no-experience actor. Higher is better.

Credit signal	Dou Dizhu	Liar's Dice	Texas Hold'em
Decision Utility (Ours)	**+0.20**	**+0.76**	**+10.30**
State Rollout	+0.12	+0.40	+1.23
Next-Game Payoff (trajectory level)	-0.07	+0.23	-2.30

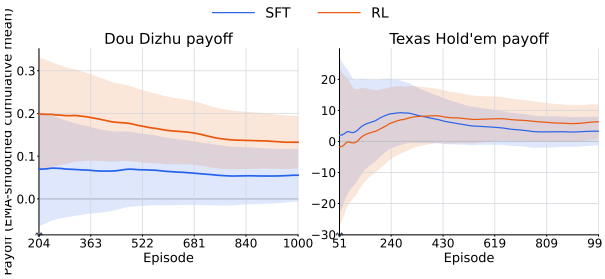
4.4 What Does Reuse-Level Optimization Change?

The learned inducer suggests transfer beyond its RL environment. EU-GRPO is trained only on Dou Dizhu, but the RL checkpoint numerically improves over SFT on Liar's Dice, Texas Hold'em, and Love Letter without further RL updates. It remains positive but does not improve over SFT on Big Two, which is also unseen during SFT. Figure 3(a) reports payoff over a long evaluation sequence after training. These results suggest that utility optimization changes the induction policy rather than only memorizing Dou Dizhu strategies, although they do not establish universal cross-environment transfer.

RL shifts experience from broad reuse to higher-utility intervention. We characterize induced experience using state-level diagnostics. *Activation coverage* measures how often experience is exposed, while *experience utility* measures its average effect on action value over all states. Figure 3(b) shows that RL reduces activation coverage from 33.19% to 25.00%, while experience utility changes from -0.5814 to $+0.5011$ and conditional utility from -1.7581 to $+2.0330$. Action divergence increases from 16.38% to 20.71%. Harmful activation also rises from 23.38% to 27.14%, so these results do not support the stronger claim that RL learns uniformly safer or more selective conditions. They instead show lower-coverage but substantially higher-utility interventions.

4.5 Applicability as an Enforced Validity Boundary

We test whether enforcing an applicability boundary can be replaced by describing it to the actor. Starting from the constructed stores, we use GPT to rewrite each condition as natural language, place the description in the actor prompt, and disable executable activation. This preserves the condition's semantic content but removes its control over exposure. The replacement reduces win rate from 85.5% to 84.0% on Liar's Dice and from 60.5% to 53.5% on Dou Dizhu. Thus, knowing when a



(a) Payoff in the long-episode evaluation after RL training.

Diagnostic metric	RL	SFT
Activation coverage	25.00%	33.19%
Experience utility	**+0.5011** [+0.1261,+0.9169]	−0.5814 [−1.2512, −0.0567]
Conditional utility	**+2.0330** [+0.6778,+3.4204]	−1.7581 [−3.6508, −0.2824]
Harmful activation	27.14%	23.38%
Action divergence	20.71%	16.38%

(b) Experience-intervention diagnostics.

Figure 3: Long-episode payoff and state-level intervention diagnostics. RL reduces exposure coverage while increasing average and conditional utility; the harmful-activation result prevents interpreting this shift as uniformly safer gating.

Table 3: Cross-environment and cross-model transfer results. Left: performance on Tide and ALFWorld. Right: cross-model paired terminal-return change (Δ) and intervention diagnostics.

(a) Cross-environment transfer

Method	Tide	ALFWorld
No-experience	0.0	44.0
ExpeL + DeepSeek-V4	0.20	56.0
EvoExp + Qwen3-8B	0.02	47.0
EvoExp + Qwen3-8B-SFT	0.13	61.0
EvoExp + Qwen3-8B-RL	**0.22**	**65.0**

(b) Cross-model transfer

Environment	Paired Δ	Retrieval rate	Action divergence
Dou Dizhu	+0.120	8.43%	19.15%
Liar's Dice	+0.307	15.17%	57.53%
Texas Hold'em	-0.167	11.31%	6.35%

lesson applies is not equivalent to asking the actor to ignore it when irrelevant: an enforced validity boundary prevents inapplicable guidance from entering the decision context in the first place.

We further evaluate explicit conditions in a 1,000-step long-horizon construction task. On Dou Dizhu, removing conditions causes the store to grow rapidly while performance declines. With executable conditions, the payoff gain remains near $+0.6$ and the store stabilizes around 30 entries. Liar's Dice shows a different failure mode: without conditions, the store fluctuates between zero and ten entries and obtains a consistently lower win rate; the condition-aware variant improves steadily while maintaining a controlled store size. These findings show that applicability is an operational validity boundary rather than a formatting choice.

4.6 Does Inducer Training Transfer Beyond Dou Dizhu?

Tide is a synthetic game whose concept and complete rules were authored by GPT rather than adapted from an existing game; its rules appear in Appendix A.6. We evaluate ALFWorld (Shridhar et al., 2021), an out-of-distribution benchmark, on a fixed 100-episode subset. Table 3 tests whether utility-based training merely fits Dou Dizhu strategies or improves the induction policy. The RL checkpoint performs best in both environments, exceeding the base inducer by 0.20 on Tide and 18 points on ALFWorld, respectively. Together with the main five-game results, these outcomes suggest cross-environment transfer of the learned induction policy. In the cross-model evaluation, the paired change is positive on Dou Dizhu and Liar's Dice, but negative on Texas Hold'em.

5 Conclusion

We presented EvoExp, a trainable experience inducer for utility-driven memory induction. EU-GRPO optimizes induction using paired differences in estimated action value for a frozen actor under current and updated memory stores across decision states. Experiments across five strategic games show improvements over the no-memory control, while evaluations on Tide and ALFWorld provide evidence that the learned induction procedure can transfer to environments excluded from training. Once trained, EvoExp constructs new memories from fresh trajectories without further parameter updates or a deployment-time value evaluator.

AI USE STATEMENT

Generative AI tools were used to assist with language editing, code drafting and debugging, and other routine research workflows. They were not treated as authoritative sources and did not autonomously determine the research questions, methodological choices, experimental protocol, or conclusions. Separately, language models are integral components of the proposed method and experimental baselines; their roles, model identities, and evaluation settings are reported in the paper. The authors reviewed all AI-assisted text, verified AI-assisted code through testing and inspection, and checked reported results against experiment records. The authors take responsibility for the final content of the paper, including all claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies learning from interaction in simulated strategic games and a standard embodied-agent benchmark; it does not involve human subjects or the collection of personal or sensitive data. The learned memories may nonetheless inherit errors or biases from their source trajectories and language models, and an incorrect applicability condition can expose misleading guidance at decision time. We also acknowledge the computational and environmental costs of model training and repeated rollout-based evaluation, and encourage reporting and minimizing these costs in future extensions.

REPRODUCIBILITY STATEMENT

We provide the algorithmic specification of EvoExp and Experience-Utility GRPO in Section 3, and describe the environments, baselines, matched evaluation protocol, model configurations, and experience-book construction procedure in Section 4.1. The appendix reports the complete EU-GRPO hyperparameters, permitted state fields and applicability-condition validator, prompt templates, state-level diagnostic metrics, and environment-specific reward, split, and failure-handling details. Formal comparisons use the same ordered source groups and held-out initial scenarios across methods, freeze experience books before evaluation, and record the updater checkpoint, decoding configuration, source-group identifier, active entries, and content hash for each book. These specifications are intended to support independent reconstruction of the training and evaluation pipeline and verification of the reported comparisons.

REFERENCES

Yishuo Cai, Xingyu Guo, Xuancheng Huang, Jinhua Du, Can Huang, Wenxuan Huang, Wenhan Ma, Yuyang Hu, Aohan Zeng, Jie Tang, and Xu Sun. From player to master: Enhancing test-time learning of llm agents via reinforcement learning over memory. 2026. URL https://arxiv.org/abs/2606.08656.

Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte carlo sampling for regret minimization in extensive games. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta (eds.), *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. URL https://proceedings.neurips.cc/paper_files/paper/2009/file/00411460f7c92d2124a67ea0f4cb5f85-Paper.pdf.

Ruoran Li, Xinghua Zhang, Haiyang Yu, Shitong Duan, Xiang Li, Wenxin Xiang, Chonghua Liao, Xudong Guo, Yongbin Li, and Jinli Suo. MemPO: Self-memory policy optimization for long-horizon agents. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23286–23301, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1166. URL https://aclanthology.org/2026.findings-acl.1166/.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. 2026:94327–94354, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

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.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL https://qwen.ai/blog?id=qwen3.8.

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

Zhiyu Shen, Ziming Wu, Fuming Lai, Shaobing Lian, and Yanghui Rao. Membuilder: Reinforcing llms for long-term memory construction via attributed dense rewards. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27868–27887, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. volume 36, 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, 2021. URL https://arxiv.org/abs/2010.03768.

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. *arXiv preprint arXiv:2305.16291*, 2023.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026a.

Wujiang Xu, Yu Wang, Kai Mei, Kaiqu Liang, Zhenting Wang, Mingyu Jin, Han Zhang, Shi-Xiong Zhang, Wenyue Hua, Sambit Sahu, et al. Memgym: a long-horizon memory environment for llm agents. *arXiv preprint arXiv:2605.20833*, 2026b.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schuetze, Volker Tresp, and Yunpu Ma. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. pp. 12805–12825, July 2026. doi: 10.18653/v1/2026.acl-long.583. URL https://aclanthology.org/2026.acl-long.583/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. pp. 21457–21483, July 2026. doi: 10.18653/v1/2026.acl-long.981. URL https://aclanthology.org/2026.acl-long.981/.

Yanwei Yue, Boci Peng, Xuanbo Fan, Jiaxin Guo, Qiankun Li, and Yan Zhang. Mem-t: Densifying rewards for long-horizon memory agents. *arXiv preprint arXiv:2601.23014*, 2026.

Daochen Zha, Jingru Xie, Wenye Ma, Sheng Zhang, Xiangru Lian, Xia Hu, and Ji Liu. Douzero: Mastering doudizhu with self-play deep reinforcement learning. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12333–12344. PMLR, 18–24 Jul 2021. URL http://proceedings.mlr.press/v139/zha21a.html.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026a.

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

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.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024.

Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In J. Platt, D. Koller, Y. Singer, and S. Roweis (eds.), *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper_files/paper/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf.

A Appendix

A.1 EU-GRPO Training Hyperparameters

We conduct the Dou Dizhu reinforcement-learning experiment using GRPO. Table 4 reports the optimization, reward-estimation, and actor-decoding configuration used to train the experience inducer.

Table 4: Dou Dizhu EU-GRPO training and rollout hyperparameters.

Hyperparameter	**Value**
Base model	Qwen3-8B
RL algorithm	GRPO
Candidate memory updates per group (G)	16
Intermediate states per candidate (N)	128
Actor rollout branches per state (K)	16
Optimization steps	150
Learning rate	1×10^{-6}
KL coefficient	0.001
Micro-batch size	1
Minibatch size	1
Optimizer	AdamW
Weight decay	0.01
Gradient clipping (maximum norm)	1.0
Updater maximum generation length	12,288 tokens
Actor temperature	0.6
Actor top-p	0.95
Actor top-k	20
Actor thinking budget	3,000 tokens

For each RL step, we sample the $N = 128$ intermediate states without replacement and distribute them as evenly as possible across 18 strata formed by the Cartesian product of role/seat, early–middle–late game stage, and whether the actor is leading or responding to the current play. At each state, we generate $K = 16$ independent actor rollout branches and score their selected actions with the fixed DouZero action-value evaluator. We first average the branch-level action values within each state and then average over all 128 states. The relative experience reward used by GRPO is obtained by comparing this aggregate under the candidate store with the corresponding aggregate under the current store, as defined in Eq. 12.

Training-time cost. Table 5 reports the average wall-clock time per optimization step and the corresponding share of the profiled runtime. Reward computation using the DouZero action-value evaluator is the dominant cost, accounting for 65.23% of the runtime, followed by candidate generation at 28.65%. Model updates, checkpointing, and other overhead together account for less than 7%.

Table 5: Average EU-GRPO wall-clock time per optimization step. Checkpointing time is amortized across the profiled steps.

Stage	Time per step (s)	Share
Candidate generation	585.74	28.65%
Reward computation (DouZero action value)	1,333.43	65.23%
Actor/reference log-probability and advantage update	119.75	5.86%
Checkpointing (amortized)	4.98	0.24%
Other overhead	0.30	0.01%

A.2 Game-Specific State Fields

Each `applicability_condition` is stored as a string containing a Python Boolean expression, for example, `game == 'blackjack' and player_total >= 17 and can_stand`. It is neither arbitrary Python code nor a separate domain-specific language. An expression may reference only pre-registered game-state fields, allow-listed keys accessed through `state[...]`, and a restricted set of functions, including `abs`, `all`, `any`, `len`, `min`, `max`, and `sum`. Read-only dictionary operations are limited to `get`, `items`, `keys`, and `values`.

Before evaluation, the expression is parsed with `ast.parse(..., mode="eval")` and checked against an AST allow-list. The validator rejects imports, arbitrary function calls, private names, and unregistered fields. Evaluation uses an empty `__builtins__` namespace, and the final value must be Boolean. A parsing, evaluation, or evidence-matching failure raises an explicit validation error and causes the corresponding update, candidate, or data sample to be rejected; failures are not converted into no-ops.

The fields below are game-specific and contain only public information or information available to the acting player. Hidden opponent cards, undealt cards, and future randomness are excluded. The complete public action history and legal-action list are supplied separately in the trajectory prompt.

Table 6: State fields exposed to the memory updater for each game.

Game	Allowed state fields
Dou Dizhu	game, player, role, is_landlord, is_farmer, teammate_position, is_leading, is_following, is_responding, hand_count, own_hand_count, landlord_hand_count, farmer_hand_counts, min_farmer_hand_count, max_farmer_hand_count, teammate_hand_count, opponent_hand_counts, min_opponent_hand_count, max_opponent_hand_count, current_claim_type, current_claim_rank_value, current_claim_player, last_move_type, last_move_rank_value, last_move_size, last_actor_role, public_history_len, legal_action_count, legal_non_pass_count, can_pass, can_claim, can_single, can_pair, can_multi_card, can_shed_two_or_more, can_shed_three
Big Two	game, player, hand_count, opponent_hand_count, is_leading, is_responding, current_claim_type, current_claim_rank_value, pass_count, can_pass, can_claim, can_single, can_pair, can_straight3, can_multi_card, can_shed_two_or_more, can_shed_three, only_pass, only_single, only_pair, only_straight3, winning_single_count
Tide Ledger	game, player, hand_count, draw_deck_count, opponent_hand_count, opponent_draw_deck_count, own_score, opponent_score, score_margin, successful_play_actions, plays_remaining, legal_action_count, can_play, must_skip, hand_values, hand_min_value
Texas Hold'em	game, player, position, street, board_count, pot, stack, opponent_stack, committed, opponent_committed, street_committed, opponent_street_committed, to_call, current_bet, hole_rank_high, hole_rank_low, hole_is_pair, hole_is_suited, hole_is_connected, made_hand_category, legal_action_types
Blackjack	game, player, phase, player_total, player_soft, player_card_count, dealer_upcard, dealer_upcard_value, shoe_count, legal_action_types, legal_action_count, can_hit, can_stand, can_double, is_initial_hand, is_hard_total, is_soft_total, is_bust_risk, public_history_len
Love Letter	game, player, phase, own_hand_count, opponent_hand_count, draw_pile_count, burn_available, public_removed_cards, public_removed_values, own_discard_total, opponent_discard_total, discard_total_margin, own_protected, opponent_protected, has_private_peek, known_opponent_card
Liar's Dice	game, player, phase, own_dice, own_dice_counts, own_dice_count, opponent_dice_count, total_dice_count, has_current_bid, current_bid_quantity, current_bid_face, current_bidder, current_bid_order_index, bids_remaining, own_matching_count, minimum_possible_actual_count

For every game, a condition is rejected if it references a field not listed in the corresponding row, accesses hidden state, raises an exception during evaluation, or fails to evaluate to `True` for at least one observed decision state.

A.3 Prompt Templates

Game Trajectory and Evaluation Action Prompt

Trajectory generation. The frozen Qwen3.8 action model receives the current game observation, the complete legal-action list, the retrieved memories, and the public action history. Hidden state and unavailable opponent information are excluded.
The system prompt is:

```
Choose one legal action for the current game state.
Use only the actions listed under Legal actions.
If retrieved memories are shown, cite only the memories that
    directly
affected the decision.
Return exactly one action in the required action format.
```

The user prompt contains:

```
{
  "game": "<game_name>",
  "observation": "<public and player-visible state>",
  "legal_actions": ["<action_0>", "...", "<action_n>"],
  "public_history": ["<previous actions>"],
  "retrieved_memories": ["<memory records>"]
}
```

The model output is parsed as one legal action. When memory attribution is enabled, the output may additionally contain the identifiers of the retrieved memories that affected the action. These action responses are used to form the trajectory evidence and are not themselves SFT targets for the memory updater.

Combined Memory Update Prompt

Memory update. The updater must summarize the supplied episode evidence and modify the current memory book only when the evidence supports a concrete, executable change. The updater may return zero, one, or multiple native tool calls.
The system prompt is:

```
You are the memory updater, separate from the frozen game action
    model.
Update the memory book using only the supplied episode evidence.
Use memory_add_batch, and memory_delete_batch.
A game-skill or failure-reflection item must contain:
  (1) an executable recommendation, and
  (2) an applicability_condition.
The applicability_condition must be a Python boolean expression
    over
the allowed public turn-state variables. It must be valid on every
supplied evidence turn and true on at least one witness turn.
Do not infer hidden state or add unsupported rules.
For a justified no-op, return no tool calls.
```

The user prompt is a canonical JSON object of the following form:

```
{
  "mode": "combined_memory_update",
  "game": "<game_name>",
  "episode": {
    "trajectory": ["<public trajectory>"],
```

```
    "condition_states": ["<visible decision states>"],
    "outcome": "<terminal outcome>",
    "memory_usage_stats": "<retrieval and usage statistics>"
  },
  "memory_book": ["<existing memories>"]
}
```

The supervised target is the updater's native tool-call completion. The environment, rather than the model, performs transaction construction, memory-type assignment, validation, and reward computation.

A.4 Limitations

EU-GRPO additionally requires a decision-utility signal in the environment used for reinforcement learning. Our transfer experiments show that such an evaluator is not required in every deployment environment, but extending utility-driven induction to domains where no suitable training-time evaluator is available remains an important direction for future work.

A.5 Definitions and Evaluation Scope of Intervention Metrics

We use five diagnostic metrics to quantify whether a memory intervention is applicable, useful, harmful, and behaviorally effective. These metrics are computed only for Texas Hold'em states. They are not pooled across games.

Let $\mathcal{S}_\ell$ denote the set of recorded public decision states in lane ℓ, where $\ell \in \{\texttt{SFT298}, \texttt{RL-step50}\}$. For each state s, $a_M(s)$ is the action selected with memory and $a_0(s)$ is the action selected with memory disabled. Let $c(s) \in \{0, 1\}$ indicate whether at least one active non-opponent memory has an `applicability_condition` that evaluates to true on s.

For each state, we reconstruct the hidden opponent cards and future deck orders. Using the same 64 hidden-state samples and the same continuation random seeds for both actions, we estimate

$$Q_M(s) = \frac{1}{J}\sum_{j=1}^{J} R(s, a_M(s), z_{s,j}), \qquad Q_0(s) = \frac{1}{J}\sum_{j=1}^{J} R(s, a_0(s), z_{s,j}), \tag{17}$$

where $J = 64$, $z_{s,j}$ is a hidden-state completion, and R is the evaluated player's signed terminal net-chip payoff under the fixed 10,000-iteration MCCFR average policy. We define $\Delta Q(s) = Q_M(s) - Q_0(s)$.

These metrics are mechanism diagnostics rather than replacements for end-to-end terminal payoff. In particular, activation coverage measures whether a memory could apply, action divergence measures behavioral change, and the Q-based utility metrics measure policy-relative local payoff change. They do not by themselves establish improvement against the live opponent distribution.

A.6 Environment and Evaluation Details

Tide Ledger environment. Tide Ledger is a two-player alternating-turn game implemented as an explicit deterministic rules environment. Each player has a private deck $[1, 1, 2, 2, 3, 3, 4, 4]$, draws five cards into the initial hand, and keeps the remaining three cards as a private draw deck. The public board contains three lanes with initial tides $+, -, +$, respectively. A legal action is $\mathrm{PLAY}(v, \ell)$, where v is a value in the current player's hand and $\ell \in \{0, 1, 2\}$ is a lane. Playing v increases the selected lane's pressure by v; if v equals the lane's previous echo, one additional pressure point is added. The acting player becomes the lane owner and the echo is updated to v. When pressure reaches the threshold 6, the accumulated pressure is awarded to the lane owner for a $+$ lane and to the opponent of the owner for a $-$ lane. The lane is then reset and its tide is flipped. After every successful play, the acting player draws one card when the private draw deck is non-empty, and the turn changes to the other player. An episode terminates after 12 successful plays or when both players have no remaining cards. If the scores are tied, the player with the smaller sum

Table 7: Definitions of the five intervention metrics.

Metric	Definition
Activation coverage	The fraction of evaluated states for which at least one active non-opponent memory is applicable: $$\mathrm{AC}_\ell = \frac{\sum_{s\in\mathcal{S}_\ell} c(s)}{|\mathcal{S}_\ell|}.$$ The denominator is all evaluated states.
Experience utility	The average memory-minus-no-memory Q-value difference over all evaluated states: $$\mathrm{EU}_\ell = \frac{1}{|\mathcal{S}_\ell|}\sum_{s\in\mathcal{S}_\ell} \Delta Q(s).$$ This measures the average utility of the memory intervention over the full state distribution.
Conditional utility	The average Q-value difference restricted to states covered by at least one applicable memory: $$\mathrm{CU}_\ell = \frac{1}{|\mathcal{S}_\ell^{\mathrm{cov}}|}\sum_{s\in\mathcal{S}_\ell^{\mathrm{cov}}} \Delta Q(s), \quad \mathcal{S}_\ell^{\mathrm{cov}} = \{s\in\mathcal{S}_\ell : c(s)=1\}.$$ If $\mathcal{S}_\ell^{\mathrm{cov}}$ is empty, the metric is reported as undefined rather than zero.
Harmful activation rate	The fraction of covered states in which the memory action has lower estimated utility than the no-memory action: $$\mathrm{HAR}_\ell = \frac{\sum_{s\in\mathcal{S}_\ell^{\mathrm{cov}}} \mathbf{1}[\Delta Q(s)<0]}{|\mathcal{S}_\ell^{\mathrm{cov}}|}.$$ This denominator is covered states only.
Action divergence	The fraction of all evaluated states in which the memory and no-memory actions differ: $$\mathrm{AD}_\ell = \frac{\sum_{s\in\mathcal{S}_\ell} \mathbf{1}[a_M(s)\neq a_0(s)]}{|\mathcal{S}_\ell|}.$$ Actions are compared using the canonical legal-action representation.

of unplayed cards wins; an exact remaining-sum tie is resolved in favor of player 1. The terminal reward is $+1$ for the winner and -1 for the loser.

Reward and metric scale. All Tide Ledger utility and payoff values are expressed in terminal zero-sum reward units. Consequently, the expected utility lies on the scale induced by terminal outcomes (approximately $[-1, 1]$). Values such as 0.02 denote an expected utility difference of 0.02, while 0.4 denotes an expected utility difference of 0.4; they are not pressure, lane-score, or card-value measurements. The Tide Ledger action-value audit used $1{,}024$ MCCFR iterations and $4{,}096$ continuation rollouts per legal action.

ALFWorld functional environment. Our ALFWorld functional environment is not a language-model-generated rule description. It is an adapter around ALFWorld's `AlfredTWEnv` TextWorld environment. Each episode is instantiated from a concrete `game.tw-pddl` file. The public split names are mapped to the underlying ALFWorld runtime splits as `train` $\rightarrow$ `train`, `valid_seen` $\rightarrow$ `eval_in_distribution`, and `eval_out_of_distribution` $\rightarrow$ `eval_out_of_distribution`. When an explicit game-file manifest is provided, reset selects one file using the episode seed, initializes `AlfredTWEnv` with that singleton file, and obtains the initial textual observation together with TextWorld's state-dependent admissible command list. At every step, the agent must execute exactly one command from this list. TextWorld returns the next observation, the terminal flag, the success flag `won`, and the next admissible command list. The episode reward is 1 when `won=True` and 0 otherwise, and the maximum episode length is 50 steps in the reported evaluations.

ALFWorld success rate and failure handling. For a complete evaluation with N registered episodes, the reported success rate is

$$\text{SuccessRate} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\{\texttt{won}_i = \texttt{True}\}.$$

An episode that exhausts the response-validation budget, violates the admissible-action constraint, or encounters an environment failure is recorded as an explicit failure. It is not converted into a successful action or a fallback trajectory; an evaluation with missing episodes is marked incomplete.

ALFWorld applicability conditions. Memory applicability conditions are constructed only from fields observable to the agent and available at the current decision point:

$$\{\texttt{step}, \texttt{max_steps}, \texttt{last_action}, \texttt{public_history_len}, \texttt{legal_action_count}\}.$$

Here, `task_type` is the first token of the visible task description (e.g., `put` or `look`), `step` is the current step counter, `last_action` is the most recent public action, `public_history_len` is the number of previous actions, and `legal_action_count` is the size of the current TextWorld admissible command list. Hidden simulator variables, expert trajectories, and task identifiers are not used as applicability-condition inputs.