COGITO, ERGO LUDO: LEARNING EXTERNAL LANGUAGE STATES FOR INTERACTIVE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning can produce effective behavior from interaction, but the knowledge it acquires is often stored implicitly in model parameters rather than as inspectable rules or strategies. We introduce the External Language-State Agent (ELSA), an LLM-agent framework for prompt-level unknown-rule interaction, where task names, hand-written rules, and demonstrations are not provided in the prompt. ELSA represents learned rules and strategies as an external language state initialized without task-specific content. During each episode, this state is fixed and conditions a language-based value function, a language-based world model, and action selection. Across episodes, post-episode reflection revises the external state, while GRPO trains the LLM to better use the evolving state from sparse outcome feedback. Across four symbolic, text-observable environments, ELSA improves over name-free prompting and static rule-given prompting controls while producing inspectable external-state artifacts. Experiments and controls show that these gains depend on iterative external-state refinement and learned use of that state, rather than static prompting or policy optimization alone. These results suggest a route toward interactive LLM agents whose task competence is accompanied by explicit, inspectable, and reusable language-level knowledge.

1 INTRODUCTION

Learning from interaction is a central goal of reinforcement learning (Sutton, 2022; Sutton & Barto, 2018), yet deep reinforcement learning agents typically encode acquired knowledge implicitly in neural parameters (Vinyals et al., 2019; Berner et al., 2019). Such agents can learn effective behavior, but what they learn is difficult to inspect, revise, or reuse. This limitation is especially salient for language-based agents: large language models (LLMs) can express intermediate reasoning in natural language (DeepSeek, 2025; Gemini, 2025), but reasoning traces alone do not provide a persistent mechanism for converting sparse interaction feedback into explicit knowledge about environment dynamics and strategy.

This paper studies whether an LLM agent can learn from interaction in a more inspectable form. Existing paradigms only partially address this goal. Conventional deep RL improves policies through rewards but usually stores task knowledge as opaque weights (Sutton & Barto, 2018). Prompted LLM agents can use pretrained knowledge and in-context reasoning (Gemini, 2025; Hu et al., 2025; Lin & Xu, 2025), but their task understanding is largely fixed at inference time. Memory and reflection mechanisms can summarize experience, but often treat stored knowledge as passive context: they may change what is placed in the prompt, without training the model to use the evolving context more effectively. What is missing is a mechanism that treats interaction-induced knowledge as a revisable external state, while also improving how the agent operationalizes that state under sparse feedback.

We study this problem in a *prompt-level unknown-rule* setting. The agent is not given task names, hand-written task rules, demonstrations, or task-specific descriptions beyond a generic interaction interface. It observes textual states, available actions, and sparse outcome feedback. This setting does not assume that the pretrained LLM has no prior knowledge; rather, it asks whether interaction can induce an external language state that improves decision-making beyond static prompting.

We introduce External Language-State Agent (ELSA), a language-based agent architecture that couples explicit knowledge construction with reinforcement learning. The central abstraction in ELSA is an external language state $\mathcal{E}_k = (\mathcal{G}_k, \Pi_k)$ initialized as empty, where $\mathcal{G}_k$ is a rulebook

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.

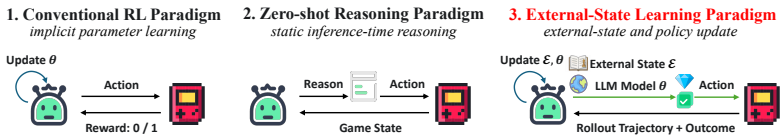


Figure 1: Comparison of interaction-learning paradigms. Conventional RL learns implicit policy parameters from reward feedback, while zero-shot LLM agents rely on inference-time reasoning without persistent task-specific learning. ELSA maintains an external language state $\mathcal{E} = (\mathcal{G}, \Pi)$ that is revised across episodes and used to condition policy learning and decision-making.

of inferred environment mechanics and Π_k is a playbook of reusable strategic heuristics. During each episode, $\mathcal{E}_k$ is fixed and conditions the shared LLM's structured decision trace, including a language-based value function (LVF) for state value evaluation, a language-based world model (LWM) (Ha & Schmidhuber, 2018; Schrittwieser et al., 2020; Hafner et al., 2025; Richens et al., 2025) for one-step consequence analysis, and the final action. Across episodes, ELSA reflects on trajectories and outcomes to revise $\mathcal{E}_k$, while sparse outcome feedback is used to improve the LLM policy through GRPO (DeepSeek, 2025).

The distinctive feature of ELSA is that the external state and the LLM policy co-evolve. Reflection updates what task knowledge is available to the agent, while GRPO updates how the model uses that knowledge to improve sparse outcomes. This distinguishes ELSA from memory-only or reflection-only agents, where a fixed policy must rely on prompted use of stored experience, and from policy optimization alone, where learned task knowledge remains implicit in the parameters. As a result, ELSA aims to learn not only effective behavior, but also explicit artifacts of what the agent has learned: rulebooks describing symbol meanings, transition dynamics, action effects, and objectives, and playbooks summarizing reusable strategies and failure-avoidance heuristics.

We evaluate ELSA on four symbolic, text-observable environments spanning logical deduction, navigation, irreversible planning, and two-player board play. ELSA improves over name-free prompting and static rule-given prompting controls while producing inspectable rules and strategies. Ablations and trained variants show that removing, freezing, or bypassing external-state refinement substantially weakens performance, indicating that ELSA's gains depend on iterative explicit-knowledge refinement rather than static prompting or policy optimization alone. Additional controls show generalization to unseen layouts and adaptation to a new game by refining the target external state while keeping the transferred policy frozen.

Our contributions are as follows:

- We study prompt-level unknown-rule interaction for LLM agents, where no task names, hand-written rules, or demonstrations are provided in the prompt, and the agent must induce task-specific knowledge from sparse interaction.
- We introduce ELSA, which represents learned rules and strategies as an external language state $\mathcal{E}_k = (\mathcal{G}_k, \Pi_k)$. This state is fixed during each episode, revised through post-episode reflection, and used to condition a language-based value function, a language-based world model, and action selection.
- We couple external-state refinement with GRPO-based policy improvement, so that reflection updates what task knowledge is available while GRPO trains the LLM to use the evolving state for decision-making. Across four symbolic, text-observable environments, ablations, trained variants, and transfer controls show that ELSA's gains depend on iterative explicit-knowledge refinement rather than static prompting or policy optimization alone.

2 Related Work

Deep RL, World Models, and Explicit Knowledge. Deep reinforcement learning has achieved strong performance in interactive environments by optimizing neural policies from reward feedback, including large-scale game-playing systems such as AlphaStar (Vinyals et al., 2019) and OpenAI Five (Berner et al., 2019). Planning and model-based methods further show that decision-making can be improved by using known rules or learned predictive models: AlphaZero combines neural networks with Monte-Carlo Tree Search (Kocsis & Szepesvári, 2006; Silver et al., 2016; 2017; 2018)

under known game rules, MuZero learns a latent dynamics model for planning without being given the transition function (Schrittwieser et al., 2020), and Dreamer learns predictive latent representations for control (Hafner et al., 2025). These methods are optimized for task performance, but the knowledge they acquire is typically represented in policy parameters, search statistics, or latent states. ELSA studies a complementary objective: whether sparse interaction can produce a revisable external language state that captures learned dynamics and strategies, while still benefiting from reward-based policy improvement.

LLM Agents for Interactive Decision Making. Recent work studies LLMs as agents that reason, plan, and act through natural language. Benchmarks and environments such as GEM (Liu et al., 2025), LMGame-Bench (Hu et al., 2025), and TextArena (Guertler et al., 2025) evaluate LLM agents in interactive tasks, exposing limitations in perception, memory, and long-horizon planning. Recent frontier LLM systems also illustrate the growing inference-time capability of LLM agents in game-like settings (Gemini, 2025). A related direction uses LLMs to synthesize policies or programs; for example, PORTAL (Xu et al., 2025) uses an LLM as a policy architect to generate behavior trees in a domain-specific language. ELSA differs from these lines in that it studies across-episode learning: task-specific knowledge is revised from the agent's own interaction traces as an external language state and then used to condition later world modeling, value estimation, and action selection.

Memory, Reflection, and Skill Accumulation. A growing line of work augments LLM agents or LLM-based decision makers with reasoning-action loops, experience summaries, self-reflection, or reusable skills (Yao et al., 2022; Madaan et al., 2023; Wang et al., 2023; Zhao et al., 2024; Shinn et al., 2024). These methods show that language can provide a useful medium for storing feedback and improving future behavior. However, many such methods store experience as unstructured or retrieved context, and the policy is not necessarily trained to use an evolving memory more effectively. ELSA instead represents learned task knowledge as an external language state $\mathcal{E}_k = (\mathcal{G}_k, \Pi_k)$: the rulebook $\mathcal{G}_k$ records inferred mechanics such as symbol meanings, transitions, action effects, and objectives, while the playbook Π_k records strategic heuristics for acting under those mechanics. This structure separates what the agent believes about the environment from how it should act, and allows the evolving state to condition different decision components rather than serving only as passive memory.

Language-Based RL and LLM Post-Training. Language has also been incorporated more directly into reinforcement learning. Natural Language Reinforcement Learning (NLRL) (Feng et al., 2024) reformulates RL objects such as value estimation and policy improvement in natural language. Separately, RL post-training methods optimize LLM behavior from preference, verifiable, or outcome feedback (DeepSeek, 2025; Shao et al., 2024; Yu et al., 2025; Sheng et al., 2024). ELSA is distinct from both directions. It does not replace reinforcement learning with a language-only formulation, nor does it rely on policy optimization alone. Instead, ELSA couples external-state refinement with GRPO-based policy improvement: reflection revises what task knowledge is available, while GRPO trains the LLM to better use the evolving external state during decision-making. This distinguishes ELSA from memory-only methods, where stored experience is used only as context, and from post-training alone, where task knowledge remains primarily implicit in the model parameters.

3 METHOD

3.1 PROBLEM SETTING AND OVERVIEW

We model interaction as a Markov Decision Process $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where the agent observes a textual state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}(s_t)$, and receives sparse feedback $r_{t+1} = \mathcal{R}(s_t, a_t)$. The transition function $\mathcal{P}(s_{t+1} \mid s_t, a_t)$ specifies the next-state probability. ELSA follows a prompt-level unknown-rule protocol: prompts provide no task names, hand-written task rules, demonstrations, or task-specific descriptions beyond a generic interaction interface. This protocol controls prompt-provided task knowledge, but does not assume that the pretrained LLM has no broad priors.

ELSA builds an external learnable language state $\mathcal{E}_k = (\mathcal{G}_k, \Pi_k)$, where $\mathcal{G}_k$ is a rulebook of inferred environment mechanics and Π_k is a playbook of reusable strategic heuristics. Both are initialized without task-specific content: $\mathcal{G}_0 = \emptyset$ and $\Pi_0 = \emptyset$. A single LLM with parameters θ instantiates all ELSA modules through role-conditioned generation. During episode k, $\mathcal{E}_k$ is fixed and conditions decision-making, i.e., $a_t \sim \pi_\theta(\cdot \mid s_t, \mathcal{E}_k)$. Across episodes, $\mathcal{E}_k$ is revised through post-episode

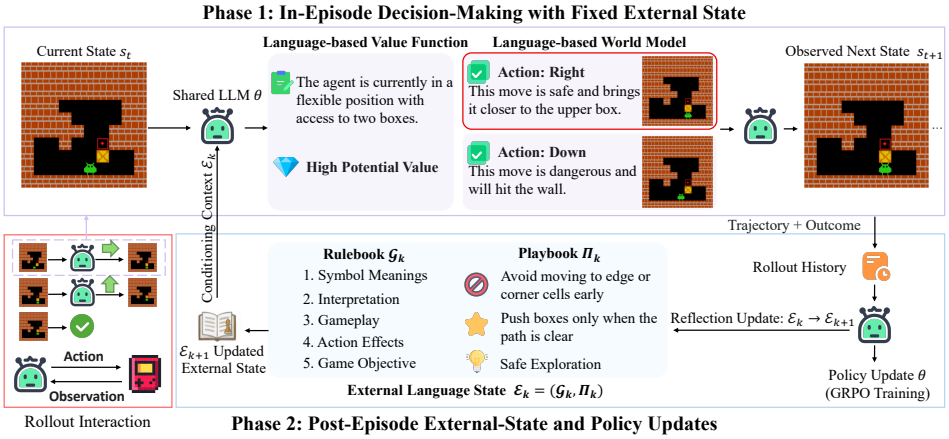


Figure 2: Overview of the External Language-State Agent (ELSA) dual learning cycle. During rollout, the current external language state $\mathcal{E} = (\mathcal{G}, \Pi)$ is fixed and conditions the shared LLM's structured decision trace, which includes a language-based value function (LVF), a language-based world model (LWM), and the final action. After rollout, the resulting trajectories and sparse outcomes drive two coupled updates: reflection revises $\mathcal{E}$, while GRPO updates the LLM parameters θ to better use the evolving external state in later decisions.

reflection, while θ is updated from sparse outcome rewards using GRPO. Thus, ELSA couples external language-state refinement with parameter-level policy improvement: the current $\mathcal{E}_k$ shapes the rollouts used for optimization, and the resulting trajectories provide evidence for revising $\mathcal{E}_k$. The next sections describe how this state is used within episodes, revised across episodes, and coupled with GRPO-based policy learning.

3.2 Phase 1: In-Episode Decision-Making with Fixed External State

During episode k, the external language state $\mathcal{E}_k$ is fixed and conditions action selection. At each step, ELSA uses a single role-conditioned LLM generation to produce a structured decision trace D_t, containing a language-based value assessment, one-step consequence predictions for candidate actions, and the final action. Conceptually, D_t instantiates two language-based decision components; the equations below describe fields within the same generation rather than separate LLM calls.

First, the LLM serves as a language-based value function (LVF). It estimates the value of the current state $\hat{v}(s_t)$ by producing a qualitative, linguistic assessment of the state's long-term potential:

$$(C_V, \hat{v}(s_t)) \sim p_\theta(\cdot \mid s_t, \mathcal{E}_k). \tag{1}$$

Here, C_V is the reasoning trace for value estimation. Unlike a conventional scalar value function, the LVF represents value in natural language, allowing ELSA to express risks, progress, strategic opportunities, and long-term potential in an inspectable form (in Figure 6 and Figure 12).

Second, within the same decision trace, the LLM serves as a language-based world model (LWM) (Ha & Schmidhuber, 2018; Hafner et al., 2025). For each candidate action $a \in \mathcal{A}(s_t)$, it predicts one-step consequences:

$$(C^a_{WM}, \hat{s}^a_{t+1}, \hat{r}^a_{t+1}) \sim p_\theta(\cdot \mid s_t, a, \mathcal{E}_k). \tag{2}$$

Here, C^a_{WM} is the reasoning trace for consequence prediction under action a, while $\hat{s}^a_{t+1}$ and $\hat{r}^a_{t+1}$ are natural-language predictions of the next state and immediate outcome. The final action $a_t \in \mathcal{A}(s_t)$ is selected in the same generation, conditioned on the current state, the LVF assessment, and the LWM predictions for candidate actions.

ELSA uses the LWM for one-step consequence analysis and action comparison (Sutton & Barto, 2018; Schrittwieser et al., 2020; Richens et al., 2025), rather than recursive imagined rollouts. After executing a_t, the environment returns the true next textual state s_{t+1}, and ELSA replans from that observed state. This keeps decision-making grounded in environment feedback while allowing the current fixed external language state $\mathcal{E}_k$ to guide each step.

3.3 PHASE 2: POST-EPISODE EXTERNAL-STATE AND POLICY UPDATES

After a reflection interval, ELSA revises its external language state using the collected interaction records and outcomes. For clarity, we describe the update using a single episode trajectory; in our experiments, the reflection input is collected over a five-episode interval. Let $\tau_k = \{(s_t, a_t, r_{t+1})\}_{t=0}^{T_k-1}$ denote the trajectory, and let z_k denote the final outcome. No ground-truth rules or demonstrations are provided to this update.

Given $\mathcal{E}_k = (\mathcal{G}_k, \Pi_k)$, the shared LLM performs post-episode reflection to propose a candidate revision $\tilde{\mathcal{E}}_{k+1} = (\tilde{\mathcal{G}}_{k+1}, \tilde{\Pi}_{k+1})$:

$$(C_k^R, \tilde{\mathcal{E}}_{k+1}) \sim p_\theta(\cdot \mid \tau_k, z_k, \mathcal{E}_k). \tag{3}$$

Here, C_k^R is the reflection trace. The proposed rulebook update revises inferred mechanics such as symbol meanings, transition patterns, action effects, and objective conditions, while the proposed playbook update distills successful and failed interaction patterns into reusable strategic heuristics.

ELSA then consolidates this candidate revision through a conservative merge:

$$\mathcal{E}_{k+1} = \mathcal{M}(\mathcal{E}_k, \tilde{\mathcal{E}}_{k+1}). \tag{4}$$

The merge operation $\mathcal{M}$ refines existing entries, resolves contradictions, and removes inconsistent or redundant content, so newly induced rules and strategies are consolidated rather than appended verbatim. This step updates the external state for future rollouts, coupling language-level knowledge refinement with the GRPO-based parameter learning described next.

3.4 GRPO-BASED POLICY IMPROVEMENT

Alongside external-state refinement, ELSA trains the shared LLM from interaction. During a training interval, rollouts are generated under the current external language state $\mathcal{E}_k$, which is used as fixed conditioning context for decision-making. Each rollout is assigned a sparse reward return determined by its final outcome. GRPO (Shao et al., 2024; DeepSeek, 2025) compares rollouts within each group to form relative advantages, and updates the LLM parameters θ on the generated decision traces and actions, favoring rollouts that perform better under the same external state.

The external language state $\mathcal{E}_k$ is not itself optimized by gradient descent; it remains explicit, revisable context updated by reflection. Crucially, ELSA does not use this state as a passive memory buffer. As $\mathcal{E}_k$ evolves, GRPO trains the LLM to better operationalize the revised rules and strategies for action selection. Thus, reflection updates what task knowledge is available, while GRPO updates how the model uses that knowledge to improve sparse outcomes. This coupling separates ELSA from memory-only or reflection-only agents, where a fixed policy uses stored experience only in context. It also separates ELSA from policy optimization alone, where learned task knowledge remains implicit in the parameters (Mnih et al., 2015; Espeholt et al., 2018).

3.5 THE AGENT'S OPERATIONAL CYCLE

Figure 2 summarizes the ELSA training cycle. Each interval starts with LLM parameters θ and external language state $\mathcal{E}_k$. ELSA collects rollouts conditioned on $\mathcal{E}_k$; the resulting trajectories and sparse outcomes are used for two coupled updates: reflection revises the external state to $\mathcal{E}_{k+1}$, while GRPO updates θ. The next interval therefore uses both revised explicit knowledge and an updated policy, completing the cycle of external-state refinement and parameter-level policy improvement.

4 EXPERIMENTS

Overview. We evaluate whether ELSA's gains arise from learning an external language state $\mathcal{E}_k = (\mathcal{G}_k, \Pi_k)$ from sparse interaction and training the LLM to use this evolving state for decision-making. Our experiments address three questions. **RQ1:** Does ELSA improve task performance under the prompt-level unknown-rule protocol while producing inspectable external-state artifacts? **RQ2:** Do the gains require iterative external-state refinement and GRPO-based learned use of that state, rather than static prompting or policy-only optimization? **RQ3:** Do ELSA's gains persist under prompt-prior, surface-symbol, and unseen-layout controls, and does frozen-weight external-state adaptation provide evidence of cross-game generalization?

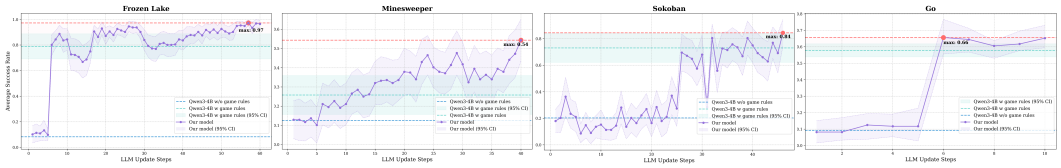


Figure 3: Training curves across FrozenLake, Minesweeper, Sokoban, and Go-like board game. The y-axis shows average success rate, and the x-axis shows update steps during GRPO training. Starting from an external language state without task-specific content, ELSA improves through interaction, external-state refinement, and policy training. Shaded regions denote 95% confidence intervals.

4.1 Experimental protocol

Setting. Our main evaluation follows a prompt-level unknown-rule protocol. Agents are not given canonical task names, hand-written task rules, demonstrations, or task-specific descriptions beyond a generic interaction interface. They observe only textual states, available actions, and sparse outcome feedback. For ELSA, the external language state $\mathcal{E}_0 = (\mathcal{G}_0, \Pi_0)$ is initialized without task-specific content. This protocol controls prompt-provided task knowledge without assuming that the pretrained LLM has no broad priors. We also include rule-given prompting as a diagnostic control to test whether static access to textual rules is sufficient.

Environments. We use four symbolic, text-observable environments: Minesweeper for logical deduction, FrozenLake for navigation and hazard avoidance, Sokoban for irreversible planning, and a Go-like 9×9 perfect-information board game that tests two-player long-horizon strategy over episodes of up to 100 moves. The agent observes raw textual states and available actions, and receives sparse outcome feedback. Detailed environment descriptions are provided in Appendix A.

Implementation and Computation. All LLM-based methods use Qwen3-4B-Instruct unless otherwise specified. For the main learning curves, each score averages 32 seeds with 8 trials per seed, yielding 256 playthroughs per environment. At each environment step, ELSA uses one LLM call to produce the structured decision trace, jointly instantiating the LVF, LWM, and final action. Every 5 episodes, it uses two additional calls for external-state refinement, one to propose rulebook/playbook revisions and one to merge them. Our goal is to study explicit knowledge acquisition and learned use of evolving external state in LLM agents, rather than to build the most compute-efficient solver for these games. Additional details are provided in Appendix B and Appendix P.

Baselines and Variants. The baselines and variants are organized by the alternative explanations they test. Name-free prompting tests whether static LLM reasoning is sufficient without interaction-based learning, while rule-given prompting tests whether static access to textual rules is sufficient. Component ablations remove or freeze external-state refinement to test whether iterative revision of $\mathcal{E}_k$ is necessary. Trained controls, including action-only GRPO and SFT / behavioral-cloning control, test whether policy-only updates or direct state-action supervision can reproduce ELSA's gains. We also report PPO as a non-LLM, state-based solver reference rather than a direct LLM-agent baseline.

4.2 Main results: task learning and external-state artifacts

We first evaluate whether ELSA can improve task performance under the prompt-level unknown-rule protocol while constructing inspectable external-state artifacts **(RQ1)**. Figure 3 reports the learning curves across the three single-agent environments and the Go-like two-player setting. Across all tasks, ELSA starts without prompt-provided task rules or task names, initializes its external language state $\mathcal{E}_0$ without task-specific content, and improves through interaction while refining $\mathcal{E}_k$ across episodes.

On Minesweeper, FrozenLake, and Sokoban, ELSA reaches approximately 54%, 97%, and 84% success, respectively, improving over name-free prompting and matching or exceeding static rule-given prompting controls. On the Go-like 9×9 game, ELSA improves from a low initial win rate and reaches 65.6%, above the static rule-given prompting control of roughly 58%. These results suggest that static access to textual rules alone does not account for ELSA's performance. Instead, ELSA benefits from refining and reusing a task-specific external language state over interaction. We inspect the learned external-state artifacts and their role in decision-making in Section 4.5. Additional baseline comparisons and backbone evaluations are provided in Appendices N and K.

4.3 ATTRIBUTION: EXTERNAL-STATE REFINEMENT AND POLICY LEARNING

We next isolate whether ELSA's gains depend on the external-state learning loop **(RQ2)**. Figure 4 compares full ELSA with variants that remove external-state refinement or induce the external state once and then keep it fixed. Without external-state refinement, learning remains weak. Inducing the rulebook and playbook once provides an initial benefit, but performance plateaus when the external state is no longer revised. In contrast, full ELSA continues to improve as its rulebook and playbook are iteratively refined across episodes. This suggests that $\mathcal{E}_k$ is not merely a static prompt prefix; its iterative refinement is part of the learning process. Matched-initialization and four-environment ablations further support the benefit of continued refinement (Appendix O).

We further test whether ELSA improves simply because it is trained with GRPO. An action-only GRPO agent uses the same backbone and RL algorithm as ELSA, but removes the LVF, LWM, and external language state. In the sparse cold-start regime, its rollout groups often have identical outcomes, especially all failures early in training, leaving little useful group-relative signal for GRPO and resulting in 0% success. This persists with probability-based valid-action sampling (Appendix E.1). A supervised fine-tuning (SFT) control, trained by behavioral cloning on $2{,}000$ expert Minesweeper state-action pairs, achieves perfect training-set accuracy but obtains 0% success on held-out evaluation layouts. These results highlight the role of interaction-induced external-state refinement: ELSA does not merely fit actions or update parameters, but learns to construct and use rules and strategies across episodes.

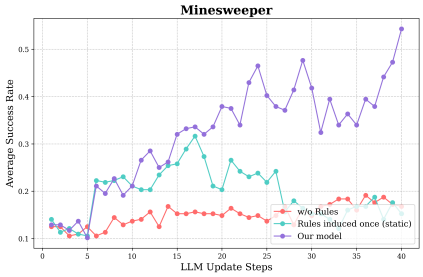


Figure 4: Ablation of external-state refinement on Minesweeper. Full ELSA (purple) continues improving as its rulebook and playbook are iteratively refined, while removing these updates (red) or freezing the induced external state after the initial update (teal) limits its sustained improvement.

We also report PPO as a non-LLM, state-based solver reference. It reaches 99% success on FrozenLake and 92% on Sokoban, compared with ELSA's 97% and 84%, respectively. ELSA focuses on acquiring and using explicit rules and strategies from sparse text-based interaction. Detailed trained-control results appear in Appendices D and E.

4.4 PRIOR CONTROLS AND CROSS-GAME GENERALIZATION

We next address **RQ3**: whether ELSA's gains can be explained by prompt-level priors, familiar surface symbols, or memorization of training layouts, and whether ELSA exhibits cross-game generalization when the LLM parameters are frozen but external-state refinement remains enabled. Table 1 reports prompt- and symbol-level controls. First, revealing the canonical game name substantially improves zero-shot prompting, confirming that task names expose useful pretrained priors. This motivates our name-free protocol: ELSA's main results are obtained without this game-name cue. Second, we replace the canonical surface symbols in each environment with randomized ASCII characters while preserving the underlying environment dynamics. ELSA's success changes only slightly under this randomized-symbol condition, from 97.3% to 97.1% on FrozenLake, from 53.5% to 51.2% on Minesweeper, and from 84.0% to 83.7% on Sokoban. This suggests that ELSA's performance is not primarily driven by recognizing familiar surface symbols. Together, these controls do not rule out all pretrained priors, but they reduce the likelihood that ELSA's gains are explained solely by game-name recall or surface-symbol recognition. Appendix F provides details on these controls.

Beyond these prior controls, we evaluate whether ELSA generalizes beyond the layouts seen during training. In the intra-game setting, we test trained agents on held-out layouts from the same environment, as summarized in Table 2. ELSA remains close to its in-domain performance, with drops of only 3.1 points on Minesweeper and 3.5 points on FrozenLake. This suggests that the learned policy and external language state are not tied only to the training seeds, but can be reused on new layouts from the same task distribution. Appendix L shows gains from broader training-layout coverage.

We further evaluate cross-game generalization through frozen-weight external-state adaptation. After training on a source environment, we transfer the LLM parameters to a target environment and freeze them. The target external language state is initialized without task-specific content, and adaptation is

Table 1: Prompt- and symbol-level controls. Entries are success rates (%). Group A compares name-free prompting with prompts that reveal canonical game names, testing whether game names expose pretrained priors. Group B replaces environment surface symbols with randomized ASCII characters while preserving the underlying dynamics, testing whether ELSA's gains are driven by familiar symbol recognition.

Method	Condition	FrozenLake	Minesweeper	Sokoban
Group A: Prompt-level Control				
Zero-shot prompting	Name-free (our setting)	10.2	12.9	20.3
Zero-shot prompting	w/ game name	38.3	20.9	48.1
Group B: Symbol-level control				
ELSA	Original symbols	97.3	53.5	84.0
ELSA	Randomized symbols	97.1	51.2	83.7

Table 2: Generalization results. Entries report success rates (%). Intra-game tests use unseen layouts; cross-game tests freeze LLM parameters and adapt only the external language state. Parentheses show gains over rule-given prompting; brackets show changes relative to in-domain performance.

Source / Policy	Cross-Environment Test		Intra-Game Unseen Layouts
	Minesweeper	FrozenLake	
Rule-given prompting	25.8	78.9	-
Minesweeper	53.5 (+27.7)	97.3 (+18.4)	50.4 [-3.1]
FrozenLake	46.9 (+21.1)	97.3 (+18.4)	93.8 [-3.5]

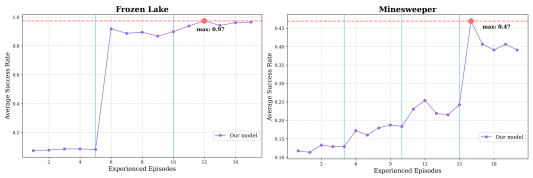


Figure 5: Frozen-weight cross-game adaptation curves. Left: Minesweeper-trained ELSA adapts to FrozenLake. Right: FrozenLake-trained ELSA adapts to Minesweeper. In both cases, only the target external language state is refined.

allowed only through reflection-based updates to $\mathcal{E}_k$. Thus, the experiment does not transfer concrete source-game rules; it tests whether the frozen source-trained policy can adapt through target-state refinement. Figure 5 shows the corresponding adaptation process. Cyan vertical lines mark external-state refinement intervals. In both directions, performance improves across reflection intervals while the LLM parameters remain frozen and only the target external state is refined. With frozen weights, a Minesweeper-trained ELSA agent reaches 97.3% on FrozenLake, while a FrozenLake-trained ELSA agent reaches 46.9% on Minesweeper (Appendix F.1).

4.5 Inspecting the learned external state and decision traces

We inspect the artifacts actually used by ELSA during rollout. These artifacts are not merely post-hoc explanations: the rulebook and playbook are components of the external language state, and the decision trace is generated during action selection. We illustrate how the learned external state conditions subsequent world modeling, value estimation, and action selection. Appendix H provides a diagnostic of decision-relevant LWM correctness, failure attribution, and external-state stability. State interventions, frozen-policy reuse, and entry-level audits further quantify the learned state's utility and quality (Appendix M).

In-Episode Decision Trace. Figure 6 shows a representative decision trace. Conditioned on the current external language state $\mathcal{E}_k$, ELSA uses the LVF to assess the current state and the LWM to compare candidate action outcomes. The selected action is therefore produced from the current observation together with task knowledge accumulated from previous interaction.

Rulebook: Environment Mechanics. Figure 7 shows an excerpt from the rulebook component $\mathcal{G}_k$ learned for Sokoban. The rulebook records inferred environment mechanics, including symbol meanings, action effects, transition patterns, and objective conditions. These rules are induced from the agent's own interaction traces rather than provided in the prompt, and are reused to condition later LWM predictions and LVF assessments.

Playbook: Strategic Heuristics. Figure 8 shows an excerpt from the playbook component Π_k. Whereas the rulebook records environment mechanics, the playbook records reusable decision heuristics distilled from interaction, such as safe exploration, constraint propagation, and avoiding irreversible failures. The playbook is revised across episodes and used as conditioning context for later value judgments and action selection. More qualitative examples are provided in Appendix Q.

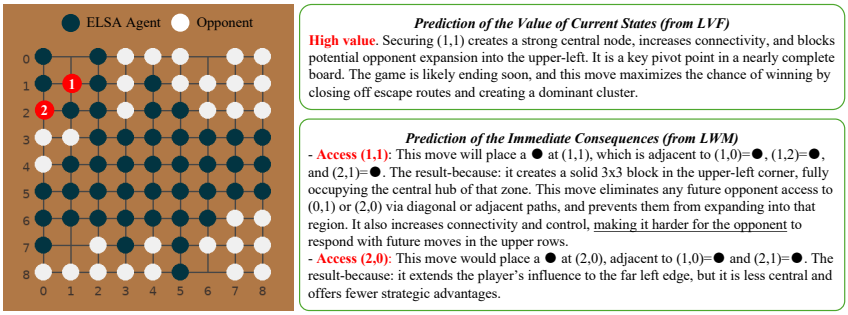


Figure 6: In-episode decision trace. Conditioned on $\mathcal{E}_k$, ELSA uses the LVF to assess the current state and the LWM to predict one-step consequences, then selects the final action.

Rule Composition	Generated-Rule Definition (Sokoban)
1. Symbol Meanings	- '#': Wall; '.': Empty passage; 'P': Player (start position) - 'B': Blue token (initially a movable box); 'G': Green token
2. Interpretation	- A valid action (UP, DOWN, LEFT, RIGHT) can only be taken from a cell adjacent to the player ('P'). - If the target cell is empty ('.'), the player moves there.
3. Gameplay	- The player controls a single 'P' that can move one cell at a time in four directions. - their final positions are determined by the solved state (all 'B' tokens on 'G' cells).
4. Action Effects	- Valid movement (player or box) updates the game state and advances the turn. - If a box is pushed into a wall ('#') or into another box, the move is invalid and no change occurs.
5. Game Objective	- **Win Condition**: All boxes ('B') are successfully moved to goal cells ('G') and marked with '√'. - **Loss Conditions**: The player exceeds the maximum number of allowed steps.

Figure 7: Excerpt from the learned rulebook $\mathcal{G}_k$ for Sokoban. The rulebook is induced from interaction traces and reused to condition LVF and LWM. The complete rulebook is provided in Figure 21.

Derived Strategic Concept		Evidence from Model (Minesweeper)
Method	Certainty	If a revealed number has exactly n adjacent unrevealed cells, those cells must contain the n mines.
	Elimination	Use numerical clues to deduce mine locations: When a number is revealed, use logical deduction to determine which unrevealed adjacent cells must contain mines.
	Constraint Propagation	Use pattern recognition and adjacent mine counts to deduce mine positions in complex configurations, especially when multiple revealed values provide overlapping constraints.
Principle	Maximum Information Gain	Avoid moving to edge or corner cells early – they have fewer neighbors and less information.
	Safe Exploration	Avoid blind exploration: Failed trajectories consistently involve random or peripheral that lead to mine exposure. Always validate safety before expanding.

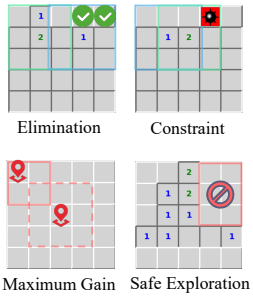


Figure 8: Excerpt from the learned playbook Π_k for Minesweeper. The playbook stores reusable decision heuristics distilled from gameplay experience.

5 Conclusion

We introduced the External Language-State Agent (ELSA), an interactive LLM-agent framework that learns external language states under prompt-level unknown-rule interaction. ELSA represents learned rules and strategies as an external language state that is fixed during each episode, revised through reflection, and used to condition a language-based value function, a one-step language-based world model, and action selection. GRPO complements this process by training the LLM to use the evolving external language state. Across symbolic, text-observable environments, ELSA improves task performance while producing inspectable external-state artifacts. Experiments and controls support that these gains depend on iterative external-state refinement rather than static prompting or policy-only optimization. Future work should extend external-state learning to larger environments and richer interaction modalities. More broadly, ELSA suggests a complementary direction for LLM-agent learning: building agents whose task competence is accompanied by explicit, inspectable, and reusable language-level knowledge.

AI USE STATEMENT

In this work, generative AI tools were used to edit the manuscript for grammar, spelling, word choice, and readability.

REFERENCES

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. DOTA 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Team DeepSeek. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Xidong Feng, Bo Liu, Yan Song, Haotian Fu, Ziyu Wan, Girish A Koushik, Zhiyuan Hu, Mengyue Yang, Ying Wen, and Jun Wang. Natural language reinforcement learning. *arXiv preprint arXiv:2411.14251*, 2024.

Team Gemini. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Leon Guertler, Bobby Cheng, Simon Yu, Bo Liu, Leshem Choshen, and Cheston Tan. TextArena. *arXiv preprint arXiv:2504.11442*, 2025.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2(3), 2018.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640:647–653, 2025.

Lanxiang Hu, Mingjia Huo, Yuxuan Zhang, Haoyang Yu, Eric P Xing, Ion Stoica, Tajana Rosing, Haojian Jin, and Hao Zhang. LMGame-Bench: How good are LLMs at playing games? *arXiv preprint arXiv:2505.15146*, 2025.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Heng Lin and Zhongwen Xu. Understanding Tool-integrated Reasoning. *arXiv preprint arXiv:2508.19201*, 2025.

Zichen Liu, Anya Sims, Keyu Duan, Changyu Chen, Diyi Yang, Wee Sun Lee, and Min Lin. GEM: A gym for generalist LLMs, 2025. URL `https://axon-rl.notion.site/gem`.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Jonathan Richens, David Abel, Alexis Bellot, and Tom Everitt. General agents need world models. *arXiv preprint arXiv:2506.01622*, 2025.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. *URL https://arxiv.org/abs/2303.11366*, 8, 2024.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. *Science*, 362(6419): 1140–1144, 2018.

Richard S Sutton. The quest for a common model of the intelligent decision maker. *arXiv preprint arXiv:2202.13252*, 2022.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An Introduction*, volume 1. MIT press Cambridge, 2018.

Sijun Tan, Michael Luo, Colin Cai, Tarun Venkat, Kyle Montgomery, Aaron Hao, Tianhao Wu, Arnav Balyan, Manan Roongta, Chenguang Wang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. rllm: A framework for post-training language agents, 2025. Notion Blog.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

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.

Zhongwen Xu, Xianliang Wang, Siyi Li, Tao Yu, Liang Wang, Qiang Fu, and Wei Yang. Agents play thousands of 3D video games. *arXiv preprint arXiv:2503.13356*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

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.

This appendix provides additional material supporting the reproducibility, attribution, and qualitative analysis of ELSA. It includes environment details, training and evaluation protocols, controls for alternative explanations, mechanism diagnostics, prompt templates, and qualitative examples of the learned external language state and decision traces.

- **Section A:** Describes the symbolic, text-observable environments and the prompt-level unknown-rule protocol used in our experiments.
- **Section B:** Reports training details, hyperparameters, LLM-call structure, external-state refinement schedule, and compute budget.
- **Section C:** Details the evaluation protocol and the computation of confidence intervals for agent performance.
- **Section D:** Presents the supervised fine-tuning (SFT) / behavioral-cloning control trained on expert state–action pairs.
- **Section E:** Studies the action-only GRPO control, which removes the LVF, LWM, and external language state, and analyzes why it fails to learn reliable policies under sparse feedback.
- **Section F:** Provides prompt-level prior and surface-symbol controls, including sanitized name-free prompts and randomized ASCII-symbol variants.
- **Section G:** Inspects ELSA's language-based value function through representative language-valued assessments.
- **Section H:** Provides a diagnostic of decision-relevant LWM correctness, failure attribution, and external-state stability.
- **Section I:** Provides failure analysis and discusses typical error modes related to external-state refinement, one-step consequence prediction, and long-horizon decision-making.
- **Section J:** Discusses the scope of ELSA and possible extensions to broader environments, model families, and agent modules.
- **Section K:** Tests whether ELSA's gains over static prompting persist across model families and sizes.
- **Section L:** Reports an additional Minesweeper experiment with expanded training seeds to assess the effect of broader training-layout coverage.
- **Section M:** Evaluates learned Rulebooks and Playbooks through state removal and replacement, reuse by a frozen base policy, and audits of correctness, coverage, and internal conflicts.
- **Section N:** Compares ELSA with memory, reflection, and reasoning-based reinforcement learning baselines, distinguishing frozen-policy methods from those that also train the policy.
- **Section O:** Measures the contributions of LVF, LWM, external-state refinement, and GRPO across four environments. Further controls examine matched initialization, corrupted rules, update frequency, and the compatibility of policy and state updates.
- **Section P:** Contains the full prompt templates for in-episode decision-making, post-episode reflection, and conservative external-state merging.
- **Section Q:** Provides qualitative examples of ELSA's behavior, including decision traces, learned rulebooks, strategic playbooks, and other external-state artifacts across environments.

A Environment Details and Prompt-Level Unknown-Rule Protocol

All game environments used in our experiments are based on TextArena (Guertler et al., 2025) and GymGo. We use symbolic, text-observable tasks so that the central challenge is not visual perception, but whether an LLM agent can infer task-specific mechanics and strategies from textual interaction and sparse outcomes. Across all environments, the main protocol hides canonical task names, hand-written rules, demonstrations, and task-specific descriptions from the prompt. The agent observes raw textual states, available actions, and sparse feedback. For ELSA, the external language state $\mathcal{E}_0 = (\mathcal{G}_0, \Pi_0)$ is initialized without task-specific content.

In the four game environments, the agent's textual observations are the raw ASCII grids provided by the environments, together with the available action list when applicable. We do not introduce any additional handcrafted interpretation or rendering layer. Thus, ELSA must infer task-specific symbol semantics from interaction rather than from prompt-provided rules.

Minesweeper is a logic puzzle where the objective is to clear a grid of all non-mine cells without detonating any mines. When a cell is revealed, it displays a number indicating how many adjacent cells contain mines, and the player must use this information to deduce mine locations. We configure Minesweeper on a 5×5 grid with 3 randomly placed mines. This task tests whether the agent can infer local symbol semantics and use them for logical deduction under sparse outcome feedback.

FrozenLake is a grid navigation task on a 6×6 grid where the agent must travel from a start tile to a goal tile while avoiding 6 randomly placed holes. We use a deterministic setting in which each action moves the agent exactly one cell in the chosen direction, removing the stochastic "slippery" transitions often associated with this environment. This focuses the evaluation on navigation, hazard avoidance, and external-state induction from text feedback.

Sokoban is a classic puzzle game where the player must push boxes to designated goal locations. The player can only push one box at a time and cannot pull boxes. This simple rule creates a large search space and requires long-horizon planning to avoid irreversible states, such as trapping a box in a corner. We use a 6×6 grid with a single box. This task tests whether the agent can learn strategic heuristics for irreversible planning from interaction.

Go-like board game is a two-player perfect-information game in which players alternately place stones and the final result depends on captured stones and controlled territory. We use a 9×9 Go-like environment with a horizon of 100 moves. ELSA plays against a rule-aware random-play opponent that samples uniformly from legal moves. The opponent is rule-aware only in the sense that it follows the legal action set; ELSA still receives only textual states, available actions, and sparse terminal feedback based on the final game outcome. This task tests two-player long-horizon strategy under sparse win/loss feedback.

B Training Details, Hyperparameters, and Compute Budget

Our implementation is based on the open-source rLLM Tan et al. (2025) framework for post-training language agents with reinforcement learning. This section details the ELSA training loop, the main hyperparameters used in our experiments, and the overall compute budget.

B.1 ELSA Training Loop

The full training loop of ELSA is summarized in Figure 9. Each training interval starts with the current LLM parameters θ and external language state $\mathcal{E}_k = (\mathcal{G}_k, \Pi_k)$, and proceeds as follows:

1. **Rollout with fixed external state.** ELSA interacts with the environment using $\mathcal{E}_k$ as fixed conditioning context. The collected rollouts contain textual observations, actions, rewards, and generated decision traces.
2. **Structured decision trace.** At each environment step, the backbone LLM produces a single structured decision trace containing the language-based value function (LVF) assessment, language-based world model (LWM) one-step predictions for candidate actions, and the final action.
3. **GRPO policy updates with fixed external state.** During each five-episode interval, the collected decision traces, actions, and sparse outcome rewards are used to update the LLM parameters θ with GRPO while $\mathcal{E}_k$ remains fixed.
4. **External-state refinement.** At the end of the interval, one proposal call uses trajectories and outcomes to propose Rulebook/Playbook revisions, and one merge call consolidates them with $\mathcal{E}_k$ into $\mathcal{E}_{k+1}$. Subsequent rollouts and policy updates use the revised state.

The next training interval therefore uses both an updated external language state and updated LLM parameters. This loop is repeated for all environments with the same architecture and training procedure.

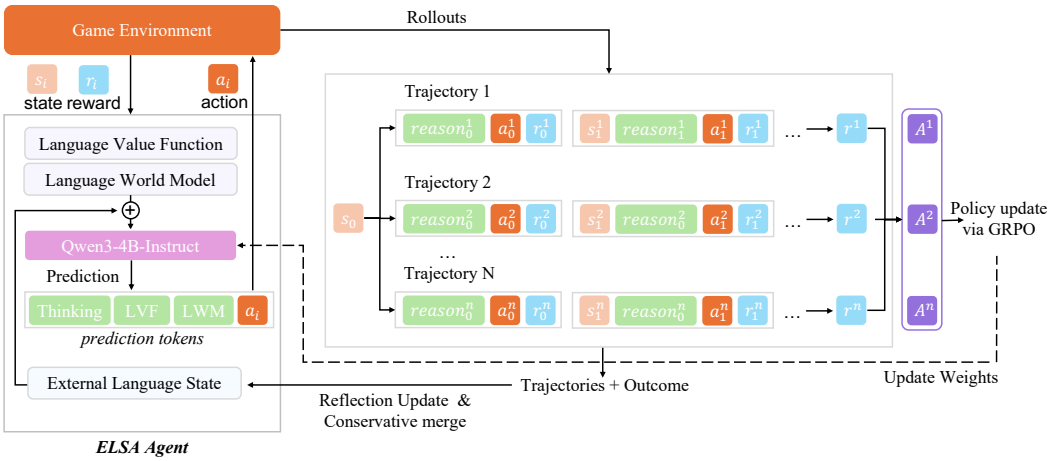


Figure 9: Overview of the ELSA training loop. Rollouts are generated with a fixed external language state $\mathcal{E}_k$, which conditions the LVF, LWM, and action selection. After rollout, reflection proposes rulebook/playbook revisions and a merge step consolidates them into $\mathcal{E}_{k+1}$, while GRPO updates the LLM parameters θ from sparse outcome feedback. No ground-truth rules or demonstrations are provided.

The proposal prompt contrasts successful and failed trajectories to extract recurring rules and strategies. The merge prompt uses the current state as its base, integrates clear and consistent revisions, consolidates duplicates, and discards ambiguous or conflicting changes. These are prompt-based criteria, not a formal guarantee of rule correctness. The fixed prompts are applied automatically without manual editing or ground-truth rules; see Figures 15 and 16.

Table 3: Main hyperparameters used for training ELSA. Unless otherwise noted, these values are shared across environments.

Category	Hyperparameter	Value
Model / Algorithm	Backbone model	Qwen3-4B-Instruct-2507
	Policy update	GRPO
	Advantage estimator	Group-relative return
External State	Initial $\mathcal{E}_0$	Empty rulebook and playbook
	Reflection interval	Every 5 episodes
	Refinement calls	Proposal + merge
Optimization	Learning rate	1×10^{-6}
	Train batch size	32
	Max gradient norm	10
	KL coefficient	0.001
Rollout / Sampling	Temperature (train / val)	0.6
	Top-p (train / val)	0.95
	Rollouts per group K	8
Length / Context	Max prompt length	8,192 tokens
	Max response length	8,192 tokens
Environment / Agent	Max episode length	Environment-specific
	Trajectory timeout	1,200 s

B.2 Hyperparameters and Compute Budget

Table 3 lists the main hyperparameters used in our experiments. Unless otherwise specified, the same settings are used across environments. At each environment step, ELSA uses one LLM call to generate the structured decision trace. Every 5 episodes, it uses two additional calls for external-state refinement: one call proposes rulebook/playbook revisions, and one call merges them into the persistent external language state.

All other low-level system and optimization settings, including gradient checkpointing, FSDP parameter and optimizer offloading, and asynchronous rollout execution, follow the default configuration of the rLLM framework and are kept fixed across experiments.

Training a single game configuration requires approximately 20 wall-clock hours on 8 GPUs, or about 160 GPU-hours. The cost scales approximately linearly with the number of sampled environment steps. This cost reflects CoT-style LLM rollouts and GRPO post-training; our goal is to study explicit knowledge acquisition and learned use of evolving external state in LLM agents, rather than to build the most compute-efficient solver for these games.

B.3 Measured Calls, Tokens, and Training Time

Table 4 separates decision calls from external-state updates. LVF, LWM, and the action are generated jointly, so they do not require three separate calls. During ordinary evaluation, both the policy and external state are frozen. The cross-game adaptation experiment instead permits target-state refinement between evaluation checkpoints, as described in Appendix F.1.

Table 4: LLM-call schedule. Decision calls are per environment step; proposal and merge calls are per five completed training episodes. Ordinary evaluation performs no external-state updates.

Phase	Method	Decision	Proposal	Merge
Training	GRPO with thinking	1	0	0
Training	Full ELSA	1	1	1
Evaluation	GRPO with thinking	1	0	0
Evaluation	Full ELSA	1	0	0

For the Minesweeper comparison in Table 5, both methods use the same interaction budget, 40 GRPO updates, and eight-GPU setup. ELSA uses longer decision inputs and outputs, and its measured end-to-end training time is approximately 20 hours versus 18 hours for GRPO with thinking. Token counts describe decision calls, not all proposal and merge generations. These measurements provide a cost comparison for this setup, rather than a claim of equal compute or a general runtime scaling law.

Table 5: Measured cost and success on Minesweeper. Input/output token counts are per decision step; time is end-to-end training time on eight GPUs.

Method	Input tokens	Output tokens	Time (h)	Success (%)
GRPO with thinking	628	1,547	$\sim$18	17.6
Full ELSA	1,187	1,823	$\sim$20	53.5

C Evaluation Protocol and Confidence Intervals

For completeness, we report the evaluation protocol and confidence-interval computation used for the learning curves with uncertainty estimates. Figure 10 reports confidence intervals for the four environments.

Evaluation protocol. After each evaluation interval, we evaluate the current ELSA checkpoint, including both the LLM parameters θ and the current external language state $\mathcal{E}_k$. No additional GRPO or reflection updates are applied during evaluation rollouts. For each environment, evaluation is performed on a fixed set of $N = 32$ level seeds, each corresponding to a different randomly generated

layout from the same game. For each seed i, we run $R = 8$ independent rollouts under the same checkpoint and compute the seed-level success rate

$$s_i = \frac{1}{R} \sum_{j=1}^{R} \mathbf{1}[\text{success}_{ij}], \qquad R = 8.$$

The scalar performance statistic at a given update step is the average seed-level success rate:

$$\hat{m} = \frac{1}{N} \sum_{i=1}^{N} s_i, \qquad N = 32.$$

The same seed-level evaluation protocol is applied to the name-free prompting control and the static rule-given prompting control.

Confidence intervals. We approximate a 95% confidence interval over evaluation seeds using the standard normal approximation,

$$\hat{m} \pm 1.96 \cdot \frac{\hat{\sigma}}{\sqrt{N}},$$

where $\hat{\sigma}$ is the sample standard deviation of $\{s_i\}_{i=1}^{N}$. In this construction, multiple rollouts on the same level seed are first aggregated into a single success rate s_i. The interval therefore reflects variability across level seeds, with rollout sampling variability aggregated within each seed.

In Figure 10, the solid purple curve denotes the mean performance of ELSA and the purple shaded band shows its 95% confidence interval. The dashed blue curve denotes the name-free prompting control, and the dashed green curve with shaded band denotes the static rule-given prompting control. Across the four environments, ELSA improves with training and surpasses the static rule-given prompting control at convergence. The uncertainty bands indicate that the observed gains are stable across evaluation seeds rather than being driven by a small number of favorable layouts.

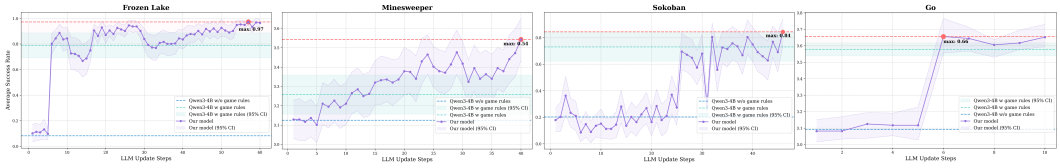


Figure 10: Average success rates with 95% confidence intervals on the four environments. The solid purple curve denotes ELSA; dashed curves denote name-free prompting and static rule-given prompting controls. Confidence intervals are computed across 32 evaluation seeds after aggregating 8 rollouts per seed.

D SFT / Behavioral-Cloning Control

To test whether ELSA's gains can be explained by supervised action fitting, we include a supervised fine-tuning (SFT) control trained by behavioral cloning on Minesweeper. The SFT control uses the same backbone as ELSA (Qwen3-4B-Instruct-2507) and is fine-tuned on $2{,}000$ expert state-action pairs. These pairs are generated by a scripted expert constructed with assistance from a stronger model (Qwen3-30B), giving the SFT control access to supervised action labels that ELSA does not receive.

The SFT control is trained with a standard cross-entropy loss using the same textual state and action representation as ELSA. It predicts actions directly from the current observation and available actions; it does not maintain an external language state $\mathcal{E}_k$, perform post-episode reflection, generate LVF/LWM decision traces, or update from interaction rewards. The model fits the supervised labels with perfect training-set action accuracy, showing that it can learn the provided state-action mapping.

However, when evaluated on the same held-out Minesweeper distribution as ELSA, the SFT control obtains 0% success on procedurally generated test boards. In contrast, ELSA receives no expert demonstrations and is trained only from sparse outcome feedback, but learns to construct and use an

external language state consisting of inferred rules and strategic heuristics. This result does not rule out richer forms of supervised training; rather, it shows that direct state-action supervision alone does not reproduce the benefits of interaction-driven external-state refinement and learned use.

E ACTION-ONLY GRPO CONTROL

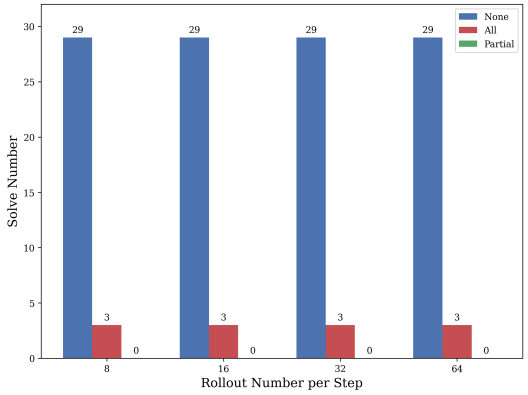


Figure 11: Distribution of within-group outcome patterns for the action-only GRPO control. "Partial" denotes mixed-outcome groups containing both successful and failed rollouts. Across group sizes from 8 to 64, sampled rollout groups contain no mixed-outcome groups. Without within-group return variation, GRPO receives little useful task-directed relative signal.

To test whether GRPO alone can train the same backbone without ELSA's external language state and structured decision trace, we consider an action-only GRPO control. This control uses the same Qwen3-4B-Instruct backbone and GRPO trainer as ELSA, but removes the external language state $\mathcal{E}_k$, the rulebook $\mathcal{G}_k$, the playbook Π_k, the language-based value function (LVF), and the language-based world model (LWM). It directly maps the current textual observation and available actions to an action.

Figure 11 analyzes the within-group outcome patterns of this action-only control. GRPO relies on relative comparisons among rollouts sampled from the same prompt or initial state. When all rollouts in a group obtain the same outcome, the group provides little task-directed relative signal for distinguishing better and worse generations. In the sparse cold-start regime, the action-only agent frequently produces homogeneous outcome groups, especially all-failure groups early in training. Increasing the group size from 8 to 64 still does not yield mixed-outcome groups in our diagnostic.

This diagnostic identifies a lack of outcome variation as a limitation of the tested action-only control. It does not establish that policy-only training must fail in general: GRPO with thinking obtains nonzero success (Appendix N.2).

E.1 PROBABILITY-BASED VALID-ACTION SAMPLING

The original action-only control already uses stochastic token-level sampling with temperature 0.6 and top-p 0.95. We additionally enumerate valid actions and sample according to their model-assigned probabilities. Table 6 shows that this increases first-action diversity, but neither sampler produces a mixed-outcome group among the 32 inspected groups, and both yield 0% success. Thus, greater first-action diversity does not resolve the absence of outcome-based relative advantages in this diagnostic. The 32 groups here are sampling-diagnostic groups, not independent training runs.

F PRIOR CONTROLS AND CROSS-GAME ADAPTATION

To assess whether ELSA's gains can be attributed to prompt-level cues or familiar surface symbols, we conduct two controls. The first compares name-free prompts with prompts that reveal canonical game names. The second randomizes Minesweeper surface symbols using ASCII characters. These

Table 6: Action-only sampling diagnostic. Mixed-outcome groups contain both successful and failed rollouts. Diversity reports the fraction of groups with different first actions.

Sampler	First-action diversity	Mixed outcomes	Success (%)
Original token sampling	81.3%	0/32	0.0
Valid-action categorical	96.9%	0/32	0.0

controls do not aim to rule out all pretrained priors in the backbone LLM; rather, they test whether the main gains can be explained by two direct sources of prior elicitation: game-name recall and familiar symbol recognition.

Sanitized, name-free prompts. At the beginning of training and evaluation under our main protocol, the agent observes only the raw text-based grid of the current state, the available actions, and generic instructions about interacting with an unknown environment. The prompts do not contain canonical game names such as "Minesweeper", "Frozen Lake", or "Sokoban", nor do they provide hand-written rules, demonstrations, or task-specific descriptions.

Name-free prompting control. The name-free prompting control uses the same sanitized prompts as ELSA, but does not maintain an external language state, perform post-episode reflection, or update its parameters from interaction. Under this setting, zero-shot Qwen3-4B obtains low success rates across the environments (Figure 3), showing that sanitized prompts alone do not elicit high task performance.

Effect of naming the game. To measure how much information is carried by the canonical game name, we add a game-name prompting control. This control is identical to the name-free prompting control except for a single sentence that names the environment, such as "You are playing Minesweeper." Revealing the game name increases zero-shot success to 20.9% on Minesweeper, 38.3% on FrozenLake, and 48.1% on Sokoban. This confirms that canonical names can expose useful pretrained priors, motivating our name-free protocol for evaluating interaction-induced external-state learning without this cue.

Randomized ASCII symbols. We further test whether ELSA relies on familiar surface symbols. We replace the canonical surface symbols with random ASCII characters while preserving the underlying environment dynamics. Under this randomized-symbol condition, ELSA's success rate decreases only slightly, from 53.5% with the original symbols to 51.2% in Minesweeper. This suggests that ELSA's performance is not primarily driven by recognizing familiar symbols.

Together, these controls do not rule out all pretrained priors. However, they reduce the likelihood that ELSA's gains are explained solely by game-name recall or surface-symbol recognition.

F.1 Cross-Game Adaptation Configurations

The two 97.3% FrozenLake scores in Table 2 arise from different policy–state configurations. The in-domain result uses a FrozenLake-trained policy θ_F and its in-domain external state $\mathcal{E}_F^{\text{in}}$. Cross-game adaptation instead freezes the Minesweeper-trained policy θ_M, starts with an empty target external state, and constructs a new FrozenLake state through target-environment interaction and reflection/merge. The source Rulebook and Playbook are not reused, and no FrozenLake GRPO updates are performed.

Table 7: FrozenLake configurations for in-domain evaluation and cross-game adaptation. Identical final scores do not imply identical policies or states.

Setting	Policy–state pair	Success (%)
FrozenLake in-domain	$(\theta_F, \mathcal{E}_F^{\text{in}})$	97.3
Minesweeper $\rightarrow$ FrozenLake, before first state update	$(\theta_M, \mathcal{E}_0)$	8.6
Minesweeper $\rightarrow$ FrozenLake, after state adaptation	$(\theta_M, \mathcal{E}_F^{\text{adapt}})$	97.3

Table 7 therefore reports post-adaptation performance, not zero-shot transfer of source-game knowledge. It demonstrates successful target-state adaptation by the frozen source-trained policy. Without

Emergent Value Pattern	Example LVF output (short excerpt)	Our analysis: qualitative value dimensions
1. Goal proximity & reachability	*"High value. The current state offers a clear path toward G… this proximity provides immediate strategic value and strong potential for further progress."*	The LVF here expresses that state value depends on distance to the goal and the existence and quality of a feasible path, not just the immediate reward. States near a solvable goal configuration are treated as intrinsically high-value.
2. Safety assessment: risk vs. robustness	*"Medium value. Moving right increases exposure to the block, while moving down keeps a safer and more flexible position for future exploration."*	The LVF here expresses a comparison of risk and future robustness between alternative actions. Value is higher for states that maintain safety and preserve many future options, and lower for states that move closer to hazards.
3. Progress vs. dead end	*"This action does not help push a block and moves the player away from it. It brings no strategic benefit and may lead to a dead end."*	The LVF here expresses that states are valuable only if they enable meaningful progress toward the objective and avoid irreversible dead ends. Reachable but non-productive positions are explicitly treated as low-value.
4. Long-horizon planning & chain value	*"Medium to low value. The block cannot be pushed directly into the goal, but a viable win sequence still exists if the player first moves behind it and then pushes it down."*	The LVF here expresses that value depends on the existence of multi-step winning plans. Even with zero immediate reward, a state remains valuable if it still supports a plausible path to victory, reflecting look-ahead over future trajectories.
5. Uncertainty & reachability under ambiguous rules	*"Value: low to medium. The player can push the adjacent block, which might open a path if the goal lies beyond it; otherwise the position is effectively stuck."*	The LVF here expresses epistemic uncertainty: value is discounted because goal reachability is unclear. The same move may either improve or ruin the position, so the state is assigned an intermediate value reflecting this uncertainty.
6. Positional control of key regions	*"High value. Controlling the central column is critical; from here the player can reach both left and right blocks, giving a strong positional advantage."*	The LVF here expresses that value can come from control of strategically important areas, not just immediate reward. Central positions that connect multiple future opportunities are treated as high-value due to their durable positional advantage.
7. High-value targets & timing	*"High value. The G and P tiles form high-value targets, and the current state offers a good opportunity to move toward them before the path becomes blocked."*	The LVF here expresses that value depends on both what high-value targets exist and when they can be reached. States are valuable when they provide a timely opportunity to approach important tiles that may later become unreachable (an opportunity window).

Figure 12: Examples of language-valued assessments produced by ELSA's LVF. Each row shows a representative LVF snippet (middle column) together with the value aspects it emphasizes (right column). These examples illustrate how the LVF expresses multiple dimensions of state quality in natural language.

an otherwise matched target-adaptation run from the untrained base policy, this comparison does not isolate the contribution of source-task training itself.

G Value in Words: Inspecting the Language-Based Value Function

The language-based value function (LVF) is generated as part of ELSA's structured decision trace rather than as a separate scalar value head or a post-hoc explanation. Given the current state and external language state $\mathcal{E}_k$, the LVF produces a qualitative assessment of the state's long-term potential in natural language. In our implementation, each LVF output is a short free-form text segment that begins with a coarse value label, followed by a brief explanation of why the state is promising, risky, or strategically useful. We treat the full text (the coarse label together with its explanation) as the LVF's value expression.

Figure 12 shows representative LVF outputs and the value aspects they emphasize. The examples illustrate that the LVF can express multiple interacting dimensions of state quality, including progress toward the goal, safety and risk, future action flexibility, and longer-horizon opportunities. Together with the LWM, these language-valued assessments form part of the decision trace used for action selection.

H Diagnostics for LWM Predictions and External-State Stability

We conduct a focused diagnostic study on Sokoban, where one-step transitions are unambiguous and failures often involve long-horizon irreversible dynamics. The goal is not to provide a complete world-model benchmark across all environments, but to inspect whether the LWM's decision-relevant predictions are locally reliable and whether the conservative merge keeps the external language state stable under repeated reflection.

Decision-relevant LWM correctness. We first inspect whether the LWM's one-step predictions match the actual environment transitions for executed actions. We use a balanced diagnostic sample of 100 decision steps from the final Sokoban checkpoint, with 50 steps from successful episodes and 50 from failed episodes. For each step, we compare the LWM prediction for the chosen action with the true next state returned by the environment. Each prediction is annotated as *correct*, *partially correct*, or *incorrect*. A prediction is marked correct if it captures the main one-step transition, such as player movement, box movement, wall collision, or invalid action status. It is marked partially correct if the main direction is right but some local details are incomplete.

Table 8: Decision-relevant LWM correctness on Sokoban. We use a balanced diagnostic sample from the final checkpoint and compare the LWM's predicted one-step consequence for the chosen action with the actual environment transition.

Subset	# decisions	Correct	Partially correct	Incorrect
Successful episodes	50	88%	8%	4%
Failed episodes	50	70%	16%	14%
Balanced sample	100	79%	12%	9%

Table 9: Manual failure attribution on Sokoban failures. Each failed episode is assigned a primary failure mode based on the external language state and decision traces near the failure point.

Primary failure mode	# episodes	Fraction
LWM prediction error	8	16%
One-step myopia / delayed deadlock	24	48%
External-state heuristic error	11	22%
Invalid/action-format issue	2	4%
Other / ambiguous	5	10%

Table 8 shows that the LWM correctly predicts the chosen action's immediate consequence in 79% of the balanced sample. The incorrect rate is higher in failed episodes (14%) than in successful episodes (4%), indicating that LWM errors can contribute to failures. At the same time, most failed-episode decisions still have correct or partially correct one-step predictions, suggesting that many failures are not caused by immediate LWM prediction errors alone.

Manual failure attribution. We next inspect 50 failed Sokoban episodes from the final checkpoint and assign each episode a primary failure mode based on the external language state and decision traces near the failure point. This analysis distinguishes immediate LWM prediction errors from failures caused by delayed irreversible consequences that are not visible from the one-step transition. When multiple factors are present, we assign the category that most directly explains the failed trajectory.

Table 9 shows that the largest failure category is one-step myopia or delayed deadlock (48%). In these cases, the immediate transition predicted by the LWM may be locally reasonable, but the selected action reduces future solvability or commits the agent to an irreversible configuration several steps later. LWM prediction errors account for 16% of inspected failures. This supports the view that the LWM is often locally useful, while Sokoban remains difficult because local one-step consequences do not always reveal delayed deadlocks.

External-state stability diagnostics. Finally, we inspect whether the merge step simply appends generated content to the external language state. We annotate proposed rulebook/playbook revisions from Sokoban reflection intervals and categorize how the conservative merge step handles each proposed entry. Each proposed entry is assigned to one primary merge behavior.

Table 10 shows that only 29% of proposed entries are accepted as new. The remaining 71% are revised, consolidated, discarded as redundant, or used to resolve conflicts. This suggests that the merge step behaves differently from append-only memory: it frequently edits, consolidates, or removes candidate entries before updating the external state.

Together, these diagnostics provide a more detailed picture of ELSA's mechanism on Sokoban. The LWM is usually locally reliable for executed one-step actions, but errors are more frequent in failed episodes. The dominant inspected failure mode is delayed deadlock and one-step myopia, while immediate LWM prediction errors account for a smaller fraction of cases. Meanwhile, the conservative merge actively consolidates the rulebook and playbook rather than appending proposed content verbatim. These diagnostics are intended as a focused case study rather than a complete world-model benchmark across all environments.

Table 10: External-state stability diagnostics on Sokoban. We annotate how proposed rulebook/playbook revisions are handled by the conservative merge step.

Merge behavior	Fraction	Interpretation
Accepted as new	29%	New useful entries retained
Revised / consolidated	37%	Candidate entries edited or merged
Discarded as redundant	26%	Prevents append-only growth
Conflict resolved	8%	Handles inconsistent entries

I Failure Analysis and External-State Error Modes

We conduct a failure analysis by manually inspecting failed episodes near the end of training, together with the external language state and decision traces used during those episodes. The goal is to understand not only whether ELSA fails, but which part of the external-state learning and decision process is responsible. We observe three recurring error modes.

Playbook overgeneralization in FrozenLake. In FrozenLake, the rulebook often correctly identifies holes as dangerous, but the playbook can overgeneralize from sparse failed trajectories into overly coarse avoidance heuristics. For example, after several failures involving holes in one region of the grid, ELSA may learn a heuristic that discourages visiting the broader region, even when some layouts require crossing it to reach the goal. In such cases, the external language state contains plausible but overly broad strategic guidance: the agent remains in safe but unproductive parts of the map and fails to commit to the unique risky path needed for success.

Long-horizon irreversible traps in Sokoban. In Sokoban, the hardest failures arise from long, irreversible action chains. The rulebook may already contain correct local mechanics, such as how boxes move, when a cell is blocked, and why cornered boxes are difficult to recover. However, a locally beneficial move, such as pushing a box closer to a goal tile or clearing a corridor, can still commit the agent to a configuration that only becomes a deadlock several steps later. In these cases, the one-step LWM/LVF decision procedure can prefer actions whose negative consequences are not visible from the immediate transition, subtly reducing the number of viable solution branches.

Risk calibration in Minesweeper. Minesweeper exhibits milder versions of external-state miscalibration. ELSA can occasionally learn plausible risk heuristics that slightly overestimate the danger of uncovering certain border cells or rare local patterns. The resulting playbook may favor low-information moves that avoid immediate risk but slow down progress, especially on larger boards where some patterns are under-observed. Here the failure is not always an incorrect rule in a single state, but an overly rigid use of a plausible heuristic that prevents the agent from gathering the evidence needed to refine it.

Taken together, these cases suggest two main limitations of the current instantiation of ELSA. First, reflection-based external-state refinement can overgeneralize from sparse trajectories, producing rules or heuristics whose scope is broader than the evidence supports. Second, one-step LVF/LWM-guided decision-making can miss rare long-horizon traps, especially in environments with irreversible dynamics. Future work could address these limitations through evidence-weighted external-state updates, uncertainty annotations on learned rules and heuristics, active exploration for under-observed situations, and deeper or more selective lookahead in domains where long-term consequences are critical.

J Scope, Scalability, and Extensions

Our current experiments focus on symbolic, text-observable environments. This section clarifies the scope of ELSA and discusses how the external-state learning loop could be composed with additional modules in broader settings. ELSA is most directly applicable when observations can be represented as compact textual or symbolic descriptions, actions are discrete or can be abstracted into high-level options, and task-relevant knowledge can be expressed as rules, strategies, constraints, or other language-level structure. In such settings, the external language state $\mathcal{E}_k = (\mathcal{G}_k, \Pi_k)$, together

with the LVF and LWM, provides a mechanism for storing, revising, and using task knowledge across episodes.

For larger or longer-horizon symbolic tasks, the high-level ELSA loop can remain the same: rollouts are conditioned on the current external state, reflection revises that state from interaction traces and outcomes, and policy learning trains the LLM to use the evolving state. However, scaling this loop requires more than increasing environment size. Larger state spaces and longer horizons would likely require more compact state abstractions, better external-state compression, stronger backbone models, and deeper or more selective lookahead. The goal is not to replace search or planning, but to provide explicit language-level knowledge that can guide them.

For partially observable or multi-agent settings, ELSA would likely need to be combined with additional state-estimation or coordination modules. For example, a belief-tracking module could summarize hidden or uncertain state information into textual context, while the external language state could store inferred constraints, conventions, role-specific strategies, or coordination heuristics. In this view, ELSA supplies an inspectable language-level memory and reasoning layer, while other modules handle perception, belief maintenance, communication, or low-level control.

For high-bandwidth visual or continuous-control environments, we do not expect the current text-only form of ELSA to operate directly on raw observations or motor commands. A more plausible path is to place ELSA above perception and control: perception modules convert raw observations into compact symbolic summaries, controllers execute low-level actions, and ELSA maintains the external language state, evaluates high-level options with the LVF/LWM, and outputs goals, subgoals, or tactical plans. Thus, ELSA is best viewed not as a general-purpose game solver, but as an external language-state layer that can make learned task knowledge explicit, revisable, and reusable within a broader agent architecture.

K Additional Backbone Evaluation

To test whether ELSA's gains depend on a single backbone, we evaluate Qwen3-4B-Instruct-2507, Qwen2.5-7B-Instruct, and Llama-3.1-8B-Instruct on Minesweeper and Sokoban. For each backbone, we compare ELSA under the prompt-level unknown-rule protocol with name-free and static rule-given prompting. Table 11 reports these comparisons across all six backbone–task combinations.

Table 11: Backbone evaluation. Entries are success rates (%). "Name-free" and "Rule-given" denote prompting controls. All backbones are instruction-tuned; the original Qwen3 and Llama Minesweeper ELSA values were reported as peak success rates.

Backbone	Task	Name-free	Rule-given	ELSA
Qwen3-4B	Minesweeper	12.9	25.8	53.5
	Sokoban	20.3	73.0	84.0
Qwen2.5-7B	Minesweeper	8.2	18.0	42.2
	Sokoban	16.4	64.8	79.3
Llama-3.1-8B	Minesweeper	7.6	17.2	38.1
	Sokoban	15.6	63.7	76.6

Full ELSA exceeds both prompting references in all six backbone–task settings. Although absolute performance varies, the improvements persist across the tested two model families and 4B–8B parameter range. These results broaden the backbone evidence without establishing a general scaling trend.

L Additional Result with Expanded Training Seeds

To test whether ELSA benefits from broader training-layout coverage, we run an additional experiment on Minesweeper with an expanded set of training seeds. We increase the number of unique training layouts from 32, used in the main experiments, to 128, while keeping the held-out evaluation protocol fixed.

Figure 13 shows the resulting learning curve. With expanded training-layout coverage, ELSA reaches a peak success rate of 62%, compared with approximately 54% in the main 32-seed setting. This provides additional evidence that ELSA is not simply fitting a small set of training layouts, but can benefit from more diverse interaction traces when refining its external language state and improving its policy.

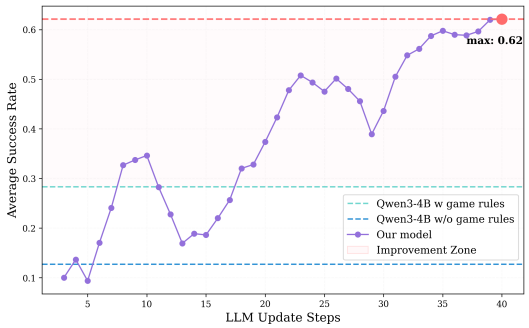


Figure 13: Learning curve for the ELSA agent on Minesweeper when trained on an expanded set of 128 unique seeds. The agent achieves a new peak success rate of 62%, surpassing the performance observed with 32 seeds.

M Utility, Reuse, and Quality of the Learned External State

We evaluate the external language state beyond qualitative examples by testing its contribution to decisions, its reuse by a policy that did not construct it, and the quality of its individual entries.

M.1 Interventions on the state with a fixed policy

We hold the final trained policy and inference settings fixed and change only the external state supplied at inference. Removing the state tests whether it remains useful after policy training. Replacing it with task-irrelevant text matched in length and format controls for additional context without the learned task-specific content. Table 12 shows that the learned state outperforms both controls in all three environments. These results support the downstream value of its task-relevant content; they do not imply that no task knowledge is also encoded in the policy parameters.

Table 12: External-state interventions with the final policy fixed. Entries are success rates (%). Only the external state supplied at inference changes.

External state	FrozenLake	Minesweeper	Sokoban
Learned state	97.3	53.5	84.0
No state	63.7	28.1	50.8
Task-irrelevant state (matched length and format)	61.3	24.2	47.3

M.2 Reuse by a frozen base policy

We provide a Minesweeper external state learned in an independent ELSA run to the frozen base Qwen policy, without task-specific parameter training of that recipient policy. Table 13 compares this condition with no external state and the static rule-given prompting baseline. The learned state improves success from 12.9% to 31.6%, exceeding the static-rule reference of 25.8%. Thus, the knowledge can benefit a policy that did not participate in its construction. This experiment tests same-task reuse across policies, whereas Appendix F.1 studies adaptation to a different game with a newly constructed target state.

Table 13: Reuse of learned knowledge by a frozen base policy on Minesweeper. The recipient policy parameters are unchanged across conditions.

Inference condition	Success (%)
No external state	12.9
Static rule-given prompting	25.8
Learned state from an independent ELSA run	31.6

M.3 Entry-level correctness, coverage, and internal consistency

We audit all 25 Rulebook entries and 31 Playbook recommendations in the final Minesweeper state. Rulebook correctness and coverage are checked against a fixed reference checklist derived from the environment specification. For the Playbook, which has no uniquely complete reference set, we assess whether recommendations are legal and consistent with the environment rules. The environment specification is used for auditing, not supplied to the agent for knowledge construction.

Rulebook entries are marked as conflicting when they give opposite conclusions under the same condition. Playbook recommendations conflict when they apply to the same situation but prescribe incompatible actions. Of the four invalid Playbook recommendations, two form a conflicting pair and two are invalid for other reasons. Table 14 reports the resulting audit. Conflict rates measure internal consistency of the final state, rather than stability over the sequence of updates. The separate Sokoban merge analysis in Appendix H describes how proposed revisions are accepted, consolidated, or discarded.

Table 14: Audit of the final Minesweeper Rulebook and Playbook. Entry-based metrics use the 25 rules or 31 recommendations, respectively; coverage is measured against reference environment facts.

Metric	Result (%)
Correct Rulebook entries	92.0
Reference environment facts covered	88.9
Valid Playbook recommendations	87.1
Rulebook entries involved in explicit conflicts	8.0
Playbook entries involved in explicit conflicts	6.5

N Comparisons with Reflection, Memory, and Reasoning Baselines

N.1 Reflection and memory with a frozen policy

We compare Reflexion (Shinn et al., 2024), ExpeL (Zhao et al., 2024), and a Voyager-style adaptation (Wang et al., 2023) on Minesweeper and Sokoban. Reflexion retains episode-level reflections, ExpeL distills reusable experience across trajectories, and Voyager-style extracts, stores, and retrieves textual strategies. The Voyager-style implementation retains the components applicable to text environments, omitting the Minecraft-specific automatic curriculum and executable code modules.

These three methods and ELSA without GRPO use the same frozen base policy with matched environment-interaction and context budgets. ELSA without GRPO continues to revise its Rulebook and Playbook. Table 15 separates these frozen-policy methods from full ELSA, which additionally trains the policy. The static rule-given baseline is shown as a reference with different prompt-provided information.

ELSA without GRPO has the highest observed success among the tested frozen-policy reflection methods on both tasks. Full ELSA further improves performance to 53.5% and 84.0%. This comparison supports the tested combination of external-state refinement and policy learning; it does not isolate the Rulebook/Playbook format from every other implementation difference between these agents.

Table 15: Reflection and memory comparisons. Success rates are reported in percent. The four reflection-only methods share a frozen base policy and matched interaction and context budgets; full ELSA additionally uses GRPO.

Method	Minesweeper	Sokoban
Static rule-given prompting (reference)	25.8	73.0
Reflection / memory with a frozen policy		
Reflexion	31.3	47.7
ExpeL	34.8	60.5
Voyager-style	29.7	56.3
ELSA without GRPO	36.3	76.2
External-state refinement and policy training		
Full ELSA	53.5	84.0

N.2 GRPO WITH THINKING

The action-only control removes both structured reasoning and the external state. To provide a stronger policy-training control, we also evaluate GRPO with thinking, which generates reasoning before acting but does not maintain or revise an external state. On Minesweeper, we use the same interaction budget, 40 GRPO updates, and eight-GPU setup for this baseline and full ELSA. GRPO with thinking reaches 17.6%, compared with 53.5% for full ELSA. Table 5 reports these results alongside token use and training time. The comparison matches interactions and update count, not total generated tokens or total compute.

O COMPONENT ABLATIONS AND UPDATE DIAGNOSTICS

O.1 COMPONENT ABLATIONS ACROSS FOUR ENVIRONMENTS

We evaluate four ablated training pipelines: removing the LVF assessment, removing the LWM one-step consequence predictions, constructing the external state once and then freezing it, and freezing the policy while retaining external-state refinement. Table 16 extends the Minesweeper analysis to all four environments. Full ELSA performs best in each environment. Freezing the state after its initial construction produces the largest drop among these ablations, while the LVF, LWM, and GRPO each contribute consistent improvements. These results indicate that iterative external-state refinement is particularly important in the tested settings.

Table 16: Component ablations across environments. Entries are success rates (%), or win rates for Go-like. "State constructed once" permits an initial construction step but no subsequent state revisions.

Setting	FrozenLake	Minesweeper	Sokoban	Go-like
Full ELSA	97.3	53.5	84.0	65.6
Without LVF	95.7	51.2	81.6	61.7
Without LWM	94.5	49.2	80.1	63.3
State constructed once, then frozen	82.8	15.2	60.9	45.3
Without GRPO (frozen policy)	87.9	36.3	76.2	54.7

O.2 CONTINUED REFINEMENT UNDER MATCHED INITIALIZATION

To isolate continued state refinement under a shared starting condition, both variants begin with the same policy checkpoint and exactly the same initial Rulebook and Playbook, and follow the same GRPO procedure. One variant freezes both books; the other revises them every five episodes. Table 17 shows a substantial benefit from continued revision. This is a separate matched-initialization experiment: its 22.7% and 54.3% scores should not be conflated with the 15.2% and 53.5% Minesweeper results in Table 16.

Table 17: Matched-initialization control on Minesweeper. Both variants start from the same policy and external state and use the same GRPO procedure.

Variant	State revisions	Success (%)
Fixed Rulebook and Playbook	None	22.7
ELSA (matched initialization)	Every five episodes	54.3

O.3 Policy and state updates within one interval

For one complete five-episode interval, we evaluate the four combinations of the policy and external state before and after their updates. Table 18 shows that either update improves performance when applied alone, and the combination yields the highest success rate. This diagnostic shows that the two updates are compatible in the inspected interval; it does not establish alignment at every training stage or a superadditive interaction between them.

Table 18: Crossed evaluation of policy and external-state updates over one five-episode interval.

Evaluated pair	Applied updates	Success (%)
$(\theta_k, \mathcal{E}_k)$	None	32.0
$(\theta_{k+1}, \mathcal{E}_k)$	Policy only	36.7
$(\theta_k, \mathcal{E}_{k+1})$	External state only	35.9
$(\theta_{k+1}, \mathcal{E}_{k+1})$	Both	39.1

O.4 Policy robustness to incorrect Rulebook entries

GRPO optimizes environment returns rather than agreement with the Rulebook. To examine behavior under misleading knowledge, we corrupt 20% of the Rulebook entries and hold this corrupted state fixed during a GRPO update. We evaluate the policy before and after the update under both the original and corrupted states. As shown in Table 19, success under the corrupted state increases from 45.3% to 49.6%, while performance after restoring the original state remains close to the pre-update result (52.3% versus 53.5%). This tests policy adaptation in the presence of incorrect knowledge, not correction of the text itself, since the state is fixed during this experiment.

Table 19: Policy robustness to Rulebook corruption on Minesweeper. Entries report success (%); policy B is updated while the corrupted state is fixed.

Policy	Original state	20%-corrupted state
A: before GRPO	53.5	45.3
B: after GRPO with corrupted state	52.3	49.6

O.5 External-state update frequency

Keeping the policy-update procedure unchanged, we compare state refinement after every episode with refinement every five episodes. The respective Minesweeper success rates are 30.1% and 53.5% (Table 20). This supports the five-episode schedule used in the main experiments. More frequent updates change the conditioning state more often and provide fewer trajectories per reflection step; they also change the number of refinement calls. The comparison therefore does not separate these possible explanations for the difference.

P Prompt Templates for Decision-Making and External-State Refinement

This section presents the core prompt templates used by ELSA. These prompts instantiate the three LLM-generation stages in our implementation: in-episode structured decision-making, post-episode

Table 20: Effect of external-state update frequency on Minesweeper. The policy update procedure is unchanged; reflection frequency and trajectory aggregation differ.

External-state update interval	Success (%)
Every episode	30.1
Every five episodes	53.5

reflection for candidate external-state revisions, and conservative merging into the persistent external language state. Figure 14 shows the in-episode decision-making prompt, Figure 15 shows the reflection prompt for proposing rulebook/playbook revisions, and Figure 16 shows the merge prompt for consolidating the proposed revisions into the external language state.

LANGUAGE-BASED WORLD MODEL THINKING PROMPT - SYSTEM

```
You are an agent playing a game on a grid, acting as a reasoning engine.

**Game Context:**
Your decisions are based on two key pieces of information:
**These rules and strategies may be incomplete or incorrect.**
- **Current Game Rules:** Your best guess of how the game works.
{game_rules}
- **Strategic Playbook:** Learned strategies and principles for achieving success.
{strategic}

**Valid Actions:**
Your only way to interact is to **access an element** on the grid. You must specify its coordinates in the format `<answer>(row, col)</answer>`.

**Instructions:**
1. **Analyze State:** Summarize the current state.
2. **Predict Long-term Value of Outcomes (Value Function Evaluation):** Evaluate the strategic value and potential of the current state for the future.
3. **Predict Immediate Consequences (World Model Simulation):** For top 2 candidate actions, predict their consequences using a "result-because" structure.
4. **Select the Best Action:** Based on the predicted consequences, choose the action that leads to the most advantageous future state.

Your response must strictly follow the format below:

<reason>
**1. Analysis of the Current State:**
[Summary of the board state.]

**2. Prediction of the Value of Current States:**
[Provide an assessment of the current state's strategic value.]
- **Value:** High value. Securing guaranteed points creates a dominant position for winning.

**3. Prediction of Immediate Consequences:**
[Analyze ONLY the top 2 candidate actions using the "result-because" structure.]
- **Access (row_A, col_A):** ...
- **Access (row_B, col_B):** ...
</reason>
<answer>(row, col)</answer>
```

LANGUAGE-BASED WORLD MODEL THINKING PROMPT - USER

```
Turn {turn}:

Observation is:
{current_observation}
```

Figure 14: Prompt template for in-episode decision-making. The LLM uses $\mathcal{E}_k$ to generate the LVF assessment, LWM predictions, and final action in one structured decision trace.

Q Qualitative External-State Artifacts and Decision Traces

We provide qualitative examples of ELSA's decision traces and learned external-state artifacts. The rulebooks and playbooks shown below are components of $\mathcal{E}_k = (\mathcal{G}_k, \Pi_k)$, and are reused to condition later LVF/LWM reasoning and action selection rather than being post-hoc explanations. We include examples of structured decision traces (Figure 17, 18), learned rulebook components (Figures 19, 20, and 21), and learned playbook components (Figures 22 and 23).

RULE AND PLAYBOOK SUMMARIZATION PROMPT - SYSTEM

```
You are a chief scientific strategist and master tactician. Your mission is to analyze extensive field data from numerous operations to distill and refine the **Master Rulebook** of a complex game.

You will be presented with a large collection of **highly successful trajectories** and **critical failure trajectories**, collected over a long period.

Your **primary task** is to perform a deep, comparative analysis to understand the fundamental principles of victory and defeat. You must act as a grand strategist, looking for universal patterns and high-level causal relationships. Your goal is to synthesize these insights to produce the **next generation's Master Rulebook**, making it more robust, accurate, and effective.

[Core Principles]
- **Think Long-Term:** Focus on universal, strategic truths that are consistently validated across many diverse scenarios. Ignore circumstantial flukes.
- **Learn from Contrast:** The most critical insights come from identifying the key strategic differences that separate winners from losers.
- **Synthesize and Consolidate:** Your output must be a single, unified, and improved Master Rulebook. Do not simply copy rules; forge a more perfect theory from all available evidence.
- **Be Authoritative and Concise:** Your rules should be stated as clear, definitive principles.

**Your output MUST be a single, consolidated `<rule>` block representing the new Master Rulebook.**

<rule>
<game_rules>
**1. Symbol Meanings:** [Define the unchanging, intrinsic properties of game elements.]
**2. Information & Interpretation:** [Define how elements reliably inform about the game state.]
**3. Gameplay & Actions:** [Define the core mechanics and interactions.]
**4. Action Effects:** [Describe the predictable outcomes of actions.]
**5. Game Objective & Termination:** [State the ultimate win/loss conditions.]
</game_rules>
<strategic>
**1. Core Strategies:** [Describe foundational, high-level strategic priorities that lead to victory.]
**2. Tactical Tips:** [List widely applicable, advantageous situational plays.]
</strategic>
</rule>
```

RULE AND PLAYBOOK SUMMARIZATION PROMPT - USER

```
**[Task]: Analyze the following successful and failed gameplays to refine the Master Rulebook.**

### Evidence File 1: Successful Trajectories (Score: 1)
{positive_trajectories_data}

### Evidence File 2: Failed Trajectories (Score: 0)
{negative_trajectories_data}

**[Instruction]: Now, based on your comparative analysis of the evidence, provide the single, updated Master Rulebook.**
```

Figure 15: Prompt template for post-episode reflection. The LLM proposes candidate rulebook and playbook revisions from recent trajectories and outcomes.

RULE AND PLAYBOOK MERGE PROMPT - SYSTEM

You are an AI assistant functioning as a Rulebook Fusion Module.
Your sole purpose is to intelligently merge two versions of a game's rulebook: an established 'Base Version' and a new 'Proposed Version'.

Your primary directive is to produce a refined 'Final Version' that prioritizes stability and logical consistency. You must operate under the assumption that the 'Base Version' is fundamentally sound and effective, while the 'Proposed Version' contains potentially valuable but unverified updates.

[Operational Principles]

1. **Prioritize the Base Version:** The 'Base Version' is your foundation. Your output must be an evolution of it, not a replacement. Preserve its structure and core principles unless there is a compelling reason to change.

2. **Be Conservative:** Adopt changes from the 'Proposed Version' with high scrutiny. Only integrate modifications or additions that represent clear, unambiguous, and essential improvements. If a proposed change is radical, confusing, or contradicts the core logic of the Base Version, you must discard it and default to the Base Version's original rule.

3. **Ensure Coherence:** The 'Final Version' must be a single, coherent, and logical document. After merging, ensure there are no redundant, conflicting, or nonsensical rules.

Your output MUST be a single, complete, and consolidated `<rule>` block representing the Final Version.

RULE AND PLAYBOOK MERGE PROMPT - USER

[Task]: Merge the following two rulebook versions into a single, updated Final Version.

Base Version (The stable, foundational rulebook)
{base_rulebook}

Proposed Version (The new draft with potential updates)
{proposed_rulebook}

[Instruction]: Now, provide the single, merged, and updated Final Version in the required format.

Figure 16: Prompt template for conservative external-state merge. The LLM consolidates candidate rulebook/playbook revisions with the existing external state, resolving contradictions and removing redundancy.

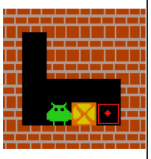
State	Prediction of the Value of Current States	Prediction of Immediate Consequences	
	High value. The player is in a direct line of action to push a box into the goal. The box is one step away from the goal, and the direction of push is clear. This is a high-probability win scenario with only one valid move required to succeed. The game is on the verge of being solved, and no further steps are needed if the push is valid.	RIGHT	The box at (4,3) will be pushed to (4,4), which is the goal ('G'). Since the cell behind the box (in the direction of movement) is empty (specifically, a goal cell), this move is valid and will mark the box as '√'. The game will immediately transition to a "Game solved" state. This move achieves the win condition in one step.
		UP	The player would move to (3,2), which is currently empty ('.'). This does not advance the box or get closer to a goal. It reduces the player's proximity to the box and creates a non-productive move. This move leads to a loss of progress and wastes a step, especially since a win is immediately available via RIGHT.

Figure 17: Example decision trace for Sokoban, including LVF assessment, LWM predictions, and final action selection.

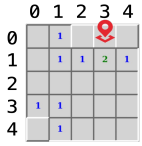
State	Prediction of the Value of Current States	Prediction of Immediate Consequences	
	The current state has a high strategic value because the only safe unrevealed cell is (0,3). Revealing it will complete the safe zones and leave only the mines unrevealed — which is acceptable in the game (as long as no mine is revealed). The game is effectively solved after (0,3) is revealed, as all other unrevealed cells are known to be mines and cannot be revealed without loss.	(0,3)	This cell is currently unrevealed and is safe (as deduced from constraints: adjacent to (1,2)=1 and (1,3)=2, and only one mine required in (0,2) and (0,3), but (0,2) is a mine, so (0,3) must be safe). Revealing (0,3) will result in a safe cell being revealed, which does not trigger a loss. This will complete the safe area and leave only known mines (at (0,2), (0,4), (4,0)) unrevealed. The game is now in a terminal state where all safe cells are revealed and no mine is exposed — thus, a win.
		(0,2)	This cell is adjacent to (1,1)=1 and is the only unrevealed neighbor. Since (1,1)=1 has exactly one mine, and (0,2) is the only unrevealed adjacent cell, it must be a mine. Revealing it would expose a mine → immediate loss.
		(0,4)	This cell is adjacent to (1,3)=2, which requires two mines. One mine is at (0,2), and (0,4) is the only other possible — so it must be a mine. Revealing it would result in a mine exposure → loss.
		(4,0)	This cell is adjacent to (3,0)=1 and is the only unrevealed neighbor → must be a mine. Revealing it → loss.

Figure 18: Example decision trace for Minesweeper, including LVF assessment, LWM predictions, and final action selection.

Rule Composition	Generated-Rule Definition (Minesweeper)
1. Symbol Meanings	- '.': Unrevealed cell (unknown content; may contain a mine or be safe). - '0': Revealed safe cell (no adjacent mines). - 'n' (where n > 0): Revealed cell with exactly 'n' adjacent mines (including diagonals). - '*': Revealed cell that contains a mine (game over condition).
2. Interpretation	- The initial state is fully obscured (all cells are '.'). - After a valid action, the game reveals the state of the selected cell. - If a revealed cell contains a mine ('*'), the game ends immediately — loss. - If a revealed cell shows a number (e.g., 1, 2), it indicates exactly how many adjacent mines exist (including diagonals), but does not reveal their specific locations. - A cell is "already revealed" if it has been previously opened and remains unchanged. - Revealing a cell with value '0' provides no direct mine information but immediately reveals all adjacent unrevealed cells (via propagation).
3. Gameplay	- On each turn, the player selects a cell (row, col) within bounds (0–4). - If the selected cell is already revealed, the action is invalid and no change occurs. - If the selected cell is unrevealed (marked '.'), the game reveals its state: - If it is a mine ('*'), the game ends — loss. - If it is a safe cell (value '0' or 'n > 0'), the cell is revealed, and the game continues. - A cell with value '0' triggers automatic revelation of all adjacent unrevealed cells (including those with '.'). - Repeated attempts to reveal already-revealed cells result in invalid actions and no change.
4. Action Effects	- Revealing a cell with value '0' causes all adjacent unrevealed cells to be automatically revealed — this is a key strategic advantage and significantly reduces the search space. - Revealing a cell with a number 'n > 0' provides partial information: exactly 'n' adjacent mines exist, but their exact positions remain unknown. - Repeated actions on already-revealed cells are invalid and do not progress the game. - Invalid actions (e.g., attempting to open a revealed cell) do not cause loss or progress the game.
5. Game Objective	- Win: All unrevealed cells are safely revealed (no mine is exposed), and the final board contains no '.' or '*'. - Lose: A mine ('*') is revealed during a valid action. - The game ends immediately upon revealing a mine. - The game is "solved" when the player has fully revealed all safe cells without exposing any mines.

Figure 19: Full learned rulebook component $\mathcal{G}_k$ for Minesweeper. Entries are induced from interaction traces.

Rule Composition	Generated-Rule Definition (FrozenLake)
1. Symbol Meanings	- 'P': Player position (starts at a given cell, moves via actions). - 'H': Hazard (indicates a wall or impassable zone; cannot move into or through). - 'G': Goal (target destination; game ends when player reaches a goal). - Empty cell ('. '): Open space; player can move into it.
2. Interpretation	- The player can only move into adjacent cells (up, down, left, right) that are not occupied by 'H' and are within the grid bounds (0â€“5 in all directions). - The game state is updated immediately after each valid action. - 'INFO: You hit a wall!' indicates an invalid move attempting to move into a cell with 'H' or outside the grid boundaries. - 'INFO: Game solved.' indicates the player has reached a goal ('G') via a valid move. - 'INFO: Game over.' indicates failure due to either falling into a hazard ('H') or exceeding the maximum number of steps (e.g., 25 turns).
3. Gameplay	- The player takes turns moving in one of four directions: UP, DOWN, LEFT, RIGHT. - Movement is only allowed to adjacent cells that are empty ('.') or contain a goal ('G'). - The player cannot move into a cell with 'H' or beyond the grid boundaries (row or column must remain within 0–5). - The game begins at the starting 'P' position and ends when the player reaches a 'G'.
4. Action Effects	- Valid move: Player shifts position to an adjacent cell if the destination is within bounds and not occupied by 'H'. - Invalid move: Movement into a cell marked 'H' or outside the grid (row or column < 0 or > 5) results in a wall hit and no positional change. - Reaching a 'G' cell ends the game with a win. - Falling into a hazard (i.e., moving into a cell marked 'H') results in immediate loss. - Exceeding the maximum number of steps (e.g., 25 turns) results in a loss.
5. Game Objective	- Objective: Reach a goal ('G') from the starting position ('P') without falling into a hazard or exceeding the maximum number of steps. - Termination conditions: - Win: Player reaches a goal ('G') via a valid move. - Loss: Player falls into a hazard ('H') or exceeds the step limit (e.g., 25 turns).

Figure 20: Learned rulebook component $\mathcal{G}_k$ for FrozenLake. Entries are induced from interaction traces.

Rule Composition	Generated-Rule Definition (Sokoban)
1. Symbol Meanings	- '#': Wall (impassable boundary) - '.': Empty passage (passable; can be traversed by the player) - 'P': Player (start position; can move to adjacent empty cells or push a box if applicable) - 'B': Blue token (initially a movable box; can be pushed if the cell behind it is empty and not occupied by a wall or another box) - 'G': Green token (goal cell; a box must be moved to a 'G' to win; once a box reaches a 'G', it is marked with '√' and cannot be moved again) - '√': Final solved state – a box has been successfully moved to a goal cell ('G'). The game is solved when all boxes ('B') are on goal cells ('G') and marked with '√'.
2. Interpretation	- A valid action (UP, DOWN, LEFT, RIGHT) can only be taken from a cell adjacent to the player ('P'). - If the target cell is empty ('.'), the player moves there. - If the target cell contains a box ('B'), the box is pushed one cell in the direction of movement **only if** the cell immediately behind the box (in that direction) is empty and not occupied by a wall ('#') or another box. - The game state is updated after each valid move. - 'INFO: Your action is valid' → move is allowed and within bounds. - 'INFO: Your action is invalid' → attempted move is out of bounds, into a wall ('#'), or into a non-empty cell that cannot be entered (e.g., pushing a box into a wall or another box). - 'INFO: Game solved' → all boxes ('B') have been successfully moved to goal cells ('G') and are marked with '√'; the game ends immediately. - 'INFO: Game over' → the maximum number of allowed steps (e.g., 15â€“16) has been reached without achieving a solved state; the player loses.
3. Gameplay	- The player controls a single 'P' that can move one cell at a time in four directions: UP, DOWN, LEFT, RIGHT. - Movement is allowed only into adjacent empty cells ('.') or to a box ('B') if the box can be pushed (i.e., the cell behind it is empty and not blocked by a wall or another box). - The player does **not** move tokens directly – instead, movement is used to either reposition the player or push a box. - Box positions ('B') are initially fixed in space; their final positions are determined by the solved state (all 'B' tokens on 'G' cells). - The objective is to guide the player' s path so that one or more boxes are pushed into goal cells ('G') and marked with '√'. - Once a box reaches a goal ('G'), it is locked in place and cannot be moved again. - The player **cannot pull** boxes – only **pushes** are allowed. - The game ends immediately upon either a win (all boxes on 'G') or a loss (step limit reached or invalid move).
4. Action Effects	- Valid movement (player or box) updates the game state and advances the turn. - If a box is successfully pushed into a goal cell ('G'), it is immediately marked with '√' and cannot be moved again. - If a box is pushed into a wall ('#') or into another box, the move is invalid and no change occurs. - Invalid movement results in no change and a clear message (e.g., "Cannot move into wall" or "Cannot push box into wall"). - Repeated invalid or looping moves (e.g., cycling between cells without progress) lead to step exhaustion and loss. - A "game solved" message appears when all boxes are on goal cells ('G') and marked with '√'. - Upon reaching a cell adjacent to a 'G', the game ends with a win **only if** a box is present on that 'G' – otherwise, the win condition remains tied to box placement.
5. Game Objective	- **Win Condition**: All boxes ('B') are successfully moved to goal cells ('G') and marked with '√'. This constitutes the final solved state. - **Loss Conditions**: - The player exceeds the maximum number of allowed steps (e.g., 15â€“16). - The player makes an invalid move (e.g., pushing a box into a wall or another box). - Boxes become stuck in positions where no further movement is possible (i.e., no path exists to any goal). - The player is unable to make any further valid moves (all boxes are blocked from reaching goals). - The game ends when either a win is achieved or a loss condition is triggered.

Figure 21: Learned rulebook component $\mathcal{G}_k$ for Sokoban. Entries are induced from interaction traces.

Derived Strategic Concept		Evidence from Model (FrozenLake)
Method	Minimize Steps	`Use efficient, direct paths to reach the goal. Avoid circular motions or backtracking.`
	Deadlock Avoidance	`If blocked, use vertical or horizontal movement to reposition - e.g., go down to a lower row, then right to reach goal.`
	Pattern Recognition	`Identify pre-existing paths or corridors (e.g., horizontal/vertical lines of '.') to exploit. Use them to move efficiently.`
Principle	Prioritize Pathfinding Over Exploration	`Always move toward the nearest goal ('G') using a clear path through open cells ('.'). Avoid detours or unnecessary exploration.`
	Constraint Satisfaction	`Avoid Hazards ('H'):** Never move into or adjacent to 'H' cells-they are impassable and lead to immediate loss.`

Figure 22: Learned playbook component Π_k for FrozenLake. The playbook stores strategic heuristics distilled from interaction.

Derived Strategic Concept		Evidence from Model (Sokoban)
Method	Repositioning	`If a box is stuck, check if the player can reposition to push it from another direction — repositioning is key to unlocking progress.`
	Deadlock Avoidance	`Avoid creating deadlocks: Never make moves that trap blocks between walls or other blocks.`
	Greedy Heuristic	`Move toward the nearest G or box closest to a G first.`
	Pattern Recognition	`Use symmetry and pattern recognition: In complex layouts, observe recurring patterns (e.g., corner formations) and exploit them to create a "clear path" to goal.`
Principle	Consequential Planning	`Plan movement paths: Always plan a route to reach a box that can be pushed, ensuring the path behind the box is clear.`
	Constraint-Based Pruning	`Never make a move that results in a box being pushed into a wall or another box — this leads to invalid moves and loss.`
	Intentional Action	`Act only when movement is valid and leads to a new, meaningful state: Every action must contribute to progress.`
	Local Optimization	`Prioritize block alignment with goal: Always aim to position a block (B or P) directly adjacent to the goal (G) before attempting to move it into it.`

Figure 23: Learned playbook component Π_k for Sokoban. The playbook stores strategic heuristics distilled from interaction.