Tencent

Cogito, Ergo Ludo: An Agent that Learns to Play by Reasoning and Planning

Sai Wang[1,2], **Yu Wu**[2] **and Zhongwen Xu**[1]

[1]Tencent, [2]Wuhan University

Abstract: The pursuit of artificial agents that can learn to master complex environments has led to remarkable successes, yet prevailing deep reinforcement learning methods often rely on immense experience, encoding their knowledge opaquely within neural network weights. We propose a different paradigm, one in which an agent learns to play by reasoning and planning. We introduce *Cogito, ergo ludo* (CEL), a novel agent architecture that leverages a Large Language Model (LLM) to build an explicit, language-based understanding of its environment's mechanics and its own strategy. Starting from a *tabula rasa* state with no prior knowledge (except action set), CEL operates on a cycle of interaction and reflection. After each episode, the agent analyzes its complete trajectory to perform two concurrent learning processes: *Rule Induction*, where it refines its explicit model of the environment's dynamics, and *Strategy and Playbook Summarization*, where it distills experiences into an actionable strategic playbook. We evaluate CEL on diverse grid-world tasks (i.e., Minesweeper, Frozen Lake, and Sokoban), and show that the CEL agent successfully learns to master these games by autonomously discovering their rules and developing effective policies from sparse rewards. Ablation studies confirm that the iterative process is critical for sustained learning. Our work demonstrates a path toward more general and interpretable agents that not only act effectively but also build a transparent and improving model of their world through explicit reasoning on raw experience.

1. Introduction

The quest to create intelligent agents [25] capable of mastering complex, interactive environments has been a long-standing goal of artificial intelligence [26]. Landmark achievements, from Deep Blue's victory in chess to AlphaGo [21–23]'s dominance in Go, have demonstrated the power of computation and search [24]. More recently, large-scale deep reinforcement learning (RL) has produced agents with superhuman abilities in complex video games [1, 28]. These systems, however, often learn inefficiently through experience, requiring immense computational resources and encoding their strategic knowledge *implicitly* within the millions of *parameters* of a neural network, rendering their decision-making processes opaque.

The advent of Large Language Models (LLMs) presents a paradigm shift, offering a new foundation for agent design grounded in reasoning and explicit knowledge representation [2, 5]. While early LLM-based agents show promise [5, 11], they often lack a structured mechanism for continuous learning and adaptation. They may operate in a zero-shot capacity [9] or rely on simple memory retrieval [5], but they do not fundamentally improve their internal model of the world's mechanics through experience. Similarly, while learned world models [8, 16, 17] have enabled agents to plan in imagined futures, their models operate on uninterpretable latent states, shrouding their "understanding" of the world in a black box. This leaves a critical gap: the need for an agent that not only acts, but truly comprehends its environment in a way that is both effective and interpretable.

Corresponding author(s): wuyucs@whu.edu.cn, zhongwenxu@tencent.com

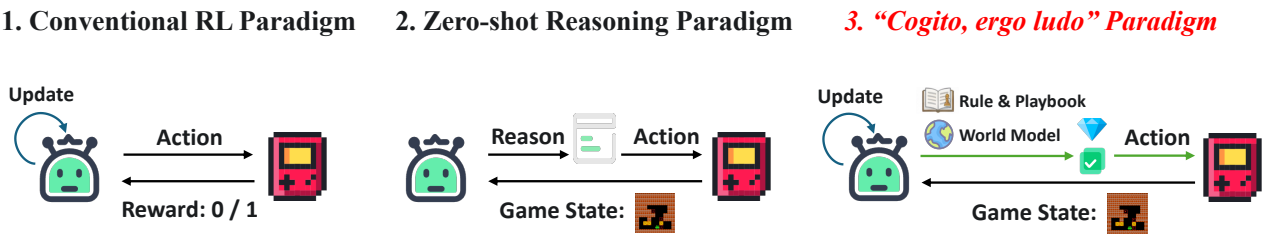


Figure 1: A Comparison of Paradigms for Game-Playing Agents. This figure contrasts three distinct agent architectures: (1) the Conventional RL Paradigm that learns an implicit policy from rewards, by updating policy weights; (2) the Zero-shot Reasoning Paradigm that leverages a static LLM model for decision-making; (3) the *Cogito, ergo ludo* Paradigm, where the agent's policy is trained by RL while a persistent knowledge base (Rule & Playbook) is built and passed *across episodes*.

In this work, we introduce *Cogito, ergo ludo* (CEL), a novel agent architecture (Figure 1) that learns to master interactive environments not just by acting, but by *reasoning* and *planning*. We propose an agent that leverages an LLM to explicitly reason about its interactions, building and refining a human-readable "world model" [7, 26] of its environment and its own strategy from the ground up. Starting from a *tabula rasa* state with no prior knowledge of the game rules, CEL learns purely through a cycle of interaction and reflection, embodying the principle of learning by thinking.

The cornerstone of CEL is its two-phase operational cycle. During an episode, the agent acts decisively by performing a lookahead search with *natural language*, using its current understanding of the world to predict the outcomes of its actions. Crucially, after each episode concludes, the agent enters a *Post-Episode Reflection* phase. In this phase, the LLM analyzes the trajectory of the preceding episode to perform two concurrent learning processes: *Rule Induction*, where it refines its explicit, language-based model of the environment's dynamics; and *Strategy and Playbook Summarization*, where it distills successful and unsuccessful patterns of behavior into an actionable strategic playbook. This refined knowledge base, both the rules of the world and the principles of how to act within it, directly informs the agent's decision-making in subsequent episodes.

We demonstrate the effectiveness of CEL across three distinct grid-world environments: the logical puzzle of Minesweeper, the navigation challenge of Frozen Lake, and the complex planning problem of Sokoban. Our experiments show that CEL successfully learns to master these tasks by autonomously discovering their rules and developing effective strategies. Ablation studies confirm that the iterative, reflective process of refining its internal knowledge is critical to its learning success. Furthermore, we provide qualitative evidence of the architecture's unique interpretability, showcasing the comprehensive, human-readable rulebooks and sophisticated strategic heuristics that it generates *entirely* from raw interaction. Our work presents a step towards agents that not only perform well, but also build a transparent and improving *understanding* of their world.

2. Related Work

Our research builds upon decades of work in artificial intelligence, drawing from and extending three key areas: the paradigm of large-scale deep reinforcement learning, the development of learned world models for planning, and the nascent field of agents driven by Large Language Models.

The Apex of Deep Reinforcement Learning. Landmark achievements such as DeepMind's AlphaStar [28] and OpenAI Five [1] demonstrated that deep reinforcement learning (RL) could attain superhuman performance in complex real-time strategy games. These systems operate at a massive scale, training for thousands of GPU-years on billions of game frames. Their strategic acumen is implicitly encoded within the weights of enormous neural networks, learned through vast experience. While immensely powerful, this approach is characterized by high sample complexity and the opaque nature of the resulting policies. Our work diverges from this paradigm by pursuing a more sample-efficient and interpretable approach, where knowledge and strategy are explicitly represented in natural language. The AlphaZero algorithm [21–23] uses the power of Monte-Carlo Tree Search (MCTS) [10] with a deep neural network, achieving superhuman performance in Chess, Shogi, and Go. But crucially, it was provided with a perfect model of the environment – the game rules, while our proposed architecture accumulates the knowledge of the game rules purely by interaction.

Planning with Learned World Models. MuZero [17] learned a latent model to predict future rewards, policies, and values, enabling effective lookahead search without being given the rules. This principle of learning and planning in imagined trajectories has been further advanced by algorithms like Dreamer [8], which learns a robust world model that allows it to master a vast suite of diverse domains, from Atari to Minecraft, with a single set of hyperparameters. Our Language-based World Model (LWM) shares this objective of predicting environmental dynamics. However, we draw a critical distinction: whereas the models in MuZero and Dreamer operate on uninterpretable latent states, our LWM is grounded in explicit, human-readable rules and transition dynamics that are themselves inferred from experience. This language symbolic foundation allows the agent to reason about and refine its understanding of the world's mechanics in natural language.

Language as the Algorithm vs. Language in the Architecture. A distinct approach is Natural Language Reinforcement Learning (NLRL) [4], which seeks to fundamentally redefine the core components of RL, such as the value function and Bellman equation, entirely within the domain of natural language. In NLRL, the value of a state is not a scalar but a descriptive text, and the policy improvement step is performed by an LLM reasoning over these linguistic value judgments. While both our approaches leverage LLMs for reasoning, our philosophy and architecture differ significantly. Rather than reformulating the RL algorithm itself into language, our framework treats the LLM as the orchestrator of a cognitive architecture composed of distinct, language-grounded modules.

LLMs as Agent Architectures. More recently, the advent of LLMs has catalyzed a new approach to agent design. Frameworks like GEM [12] and LMGame-Bench [9] provide environments and harnesses to evaluate LLM agents, highlighting challenges in perception, memory, and long-horizon planning. Gemini 2.5 Pro [5] showcases its success in complete Pokémon game playing, demonstrating the strong zero-shot reasoning abilities of the frontier LLMs. A particularly relevant approach is PORTAL [30], which uses an LLM as a "policy architect" to generate behavior trees in a domain-specific language. Unlike PORTAL, our method uses LLM as the core for planning, acting and accumulating knowledge, where the LLM directly interacts with environments.

Our work builds upon this foundation but proposes a more comprehensive cognitive architecture. Our agent learns and maintains a suite of distinct, yet interconnected, cognitive components: an explicit world model of environmental dynamics, a set of game rules, a strategic playbook, and a language-based value function. The cornerstone of our method is the post-episode reflection phase, where the LLM analyzes interaction trajectories to iteratively and simultaneously refine both its understanding of the world's rules and its own strategic playbook. This creates a cycle of self-improvement that is explicit, interpretable,

and broadly applicable to any interactive environment.

3. Method

We model the agent's interaction with its environment as a Markov Decision Process (MDP) [14], formally defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. In this framework, $\mathcal{S}$ represents the set of states, $\mathcal{A}$ the set of actions, and $\gamma \in [0, 1]$ the discount factor. The state transition function $\mathcal{P}(s_{t+1}|s_t, a_t)$ specifies the probability of transitioning to state s_{t+1} from state s_t upon taking action a_t. The reward function $\mathcal{R}(s_t, a_t)$ yields an immediate reward r_{t+1}. The agent's objective is to learn a policy $\pi(a|s)$, that maximizes the expected discounted return, defined as $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ [26].

Our central methodological contribution is to employ a single Large Language Model (LLM), denoted $\mathcal{L}$, to instantiate and manage all of the agent's cognitive functions. Our framework moves beyond the static zero-shot paradigm by continuously training an LLM based on the outcomes of the agent's interactions, allowing it to improve its core reasoning and planning capabilities over time. We represent all information pertaining to the interaction: states (s), actions (a), rewards (r), inferred environmental dynamics ($\mathcal{G}$), and strategic guidelines (Π), as natural language strings. The agent's learning unfolds over a series of episodes, indexed by k, where each episode consists of discrete time steps, indexed by t. The LLM's reasoning process is made explicit through a chain-of-thought [29], which we denote by C. By reasoning and planning, the CEL agent $\mathcal{L}$ learns to interact with the environment and maximize its rewards.

3.1. Language-based World Model

The LLM functions as a world model [7, 8], tasked with predicting the dynamics of the environment. Given the current state s_t and a candidate action a_t, the world model forecasts the subsequent state $\hat{s}_{t+1}$ and immediate reward $\hat{r}_{t+1}$. This prediction is conditioned on the agent state s_t, a potential action a_t, and the agent's current understanding of the environment's rules, $\mathcal{G}_k$. The model first generates a reasoning trace, C_{WM} (for **W**orld **M**odel), before outputting its predictions:

$$(C_{WM}, \hat{s}_{t+1}, \hat{r}_{t+1}) \sim p_{\mathcal{L}}(\cdot|s_t, a_t, \mathcal{G}_k), \tag{1}$$

where $p_{\mathcal{L}}$ is the probability distribution over text sequences generated by the LLM. Critically, the outputs $\hat{s}_{t+1}$ and $\hat{r}_{t+1}$ are not structured data types (e.g., a state tensor or a scalar value) but are descriptive natural language strings, as illustrated in Figure 5.

This predictive capability is the foundation for explicit planning [26]. By querying the world model for each potential action, the agent can simulate and evaluate a set of possible future outcomes, a process that is central to its decision-making [16, 17].

3.2. Induction of Environmental Dynamics

Following each episode $k-1$, the agent enters a reflective phase to refine its understanding of the environment's mechanics. The LLM performs rule induction by analyzing the trajectory of the concluded episode, τ_{k-1}, in light of its previously held rules, $\mathcal{G}_{k-1}$. A trajectory is the sequence of state-action-reward tuples recorded during the episode:

$$\tau_{k-1} = \{(s_0, a_0, r_1), (s_1, a_1, r_2), \ldots, (s_{T_{k-1}-1}, a_{T_{k-1}-1}, r_{T_{k-1}})\}. \tag{2}$$

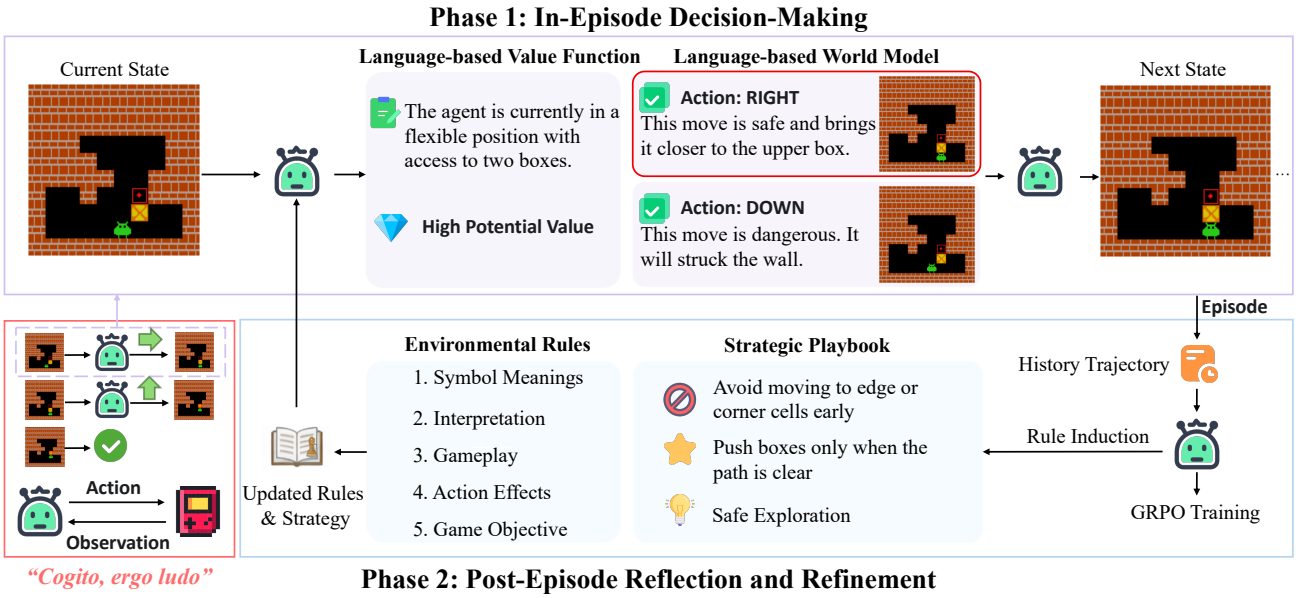


Figure 2: An overview of the *Cogito, ergo ludo* (CEL) agent's two-phase operational cycle. In Phase 1, the agent leverages its Language-based World Model (LWM) to predict the outcomes of potential actions and its Language-based Value Function (LVF) to evaluate the desirability of the resulting states, ultimately selecting the optimal action. In Phase 2, it reflects on the episode's trajectory to update its explicit knowledge base (Environmental Rules and Strategic Playbook). The agent continuously improves through this dual learning loop, which not only refines its explicit knowledge but also trains the LLM's internal parameters based on the final outcome.

The LLM processes this experiential data to generate an updated, more accurate set of rules $\mathcal{G}_k$:

$$(C_{\mathcal{G}}, \mathcal{G}_k) \sim p_{\mathcal{L}}(\cdot|\tau_{k-1}, \mathcal{G}_{k-1}), \tag{3}$$

where $C_{\mathcal{G}}$ is the reasoning trace for updating the environment's **G**overning dynamics (or **G**ame rules in game environments). We assume the agent begins with no prior knowledge (i.e., *tabula rasa*); the initial rule set $\mathcal{G}_0$ is empty, and all subsequent knowledge is derived purely from interaction [20].

3.3. Strategy and Playbook Summarization

In parallel with rule induction, the agent updates its high-level strategy. After episode $k-1$, the LLM synthesizes the trajectory τ_{k-1} and the final outcome Z_{k-1} (e.g., success/failure, final score) to update a strategic playbook, Π_k. This process distills successful and unsuccessful patterns of interaction into explicit, actionable advice:

$$(C_{\Pi}, \Pi_k) \sim p_{\mathcal{L}}(\cdot|\tau_{k-1}, Z_{k-1}, \Pi_{k-1}), \tag{4}$$

where C_{Π} is the reasoning trace for the **P**laybook update, and Π_0 is initialized with a general-purpose prompt.

This mechanism contrasts sharply with conventional reinforcement learning [3, 13], where strategy is implicitly encoded within the weights of a neural network. In such systems, adaptation is often slow,

sample-inefficient (e.g., requiring millions of interaction frames [3, 13]), and opaque. Our approach externalizes strategy into an explicit, interpretable text playbook. Insights from a single episode can be immediately incorporated into the agent's prompt context for the subsequent episode. This mechanism facilitates rapid *in-context learning*, dramatically accelerating strategic adaptation.

3.4. Language-based Value Function

To guide its planning, the agent employs the LLM $\mathcal{L}$ as a language-based value function. This component estimates the value of a state $\hat{v}(s_t)$, by providing a qualitative, linguistic assessment of the long-term potential for success from that state. This evaluation is conditioned on both the current environmental rules $\mathcal{G}_k$ and the strategic playbook Π_k:

$$(C_V, \hat{v}(s_t)) \sim p_{\mathcal{L}}(\cdot|s_t, \mathcal{G}_k, \Pi_k). \tag{5}$$

Here, C_V is the reasoning trace for the **V**alue estimation. This function provides the agent with a crucial heuristic by assessing the current state's long-term potential, which is essential for effective planning.

3.5. The Agent's Operational Cycle

The agent's operation is structured as a cyclical pipeline that alternates between two phases: in-episode decision-making and post-episode reflection (Figure 2). This architecture decouples rapid, step-by-step action selection from a more deliberate, offline knowledge consolidation process, with the LLM orchestrating both.

Phase 1: In-Episode Decision-Making. During an episode k, the agent operates with a fixed set of environmental rules $\mathcal{G}_k$ and a strategic playbook Π_k. At each time step t, it performs a structured reasoning process to select an action. First, the agent's *Language-based Value Function* (LVF) assesses the desirability of the current state, s_t, providing a high-level, holistic evaluation of its strategic potential. Concurrently, for each available action a, the agent's *Language-based World Model* (LWM) performs a one-step lookahead search to simulate the resulting state $\hat{s}_{t+1}$ and reward $\hat{r}_{t+1}$. The agent then commits to the action that the LWM predicts will lead to the most favorable outcome. The resulting (s_t, a_t, r_{t+1}) tuple is then recorded in the episode's trajectory τ_k.

Phase 2: Post-Episode Reflection and Refinement. Once an episode concludes, the agent enters the reflection phase to update its internal knowledge base. It performs *Rule Induction* by providing the LLM with the complete trajectory τ_k and the prior rule set $\mathcal{G}_k$ to produce a refined set of rules $\mathcal{G}_{k+1}$. Concurrently, it engages in *Strategy and Playbook Summarization*, where the LLM processes τ_k and the final outcome Z_k to distill key lessons, updating the playbook to Π_{k+1}. The episode's final outcome (e.g., success or failure) provides a reward signal that is used to train the agent's core LLM, making it progressively more effective at planning and strategic reasoning in future episodes.

This two-phase cycle enables the agent to both act decisively on its current understanding and systematically improve its environmental model and strategic acumen.

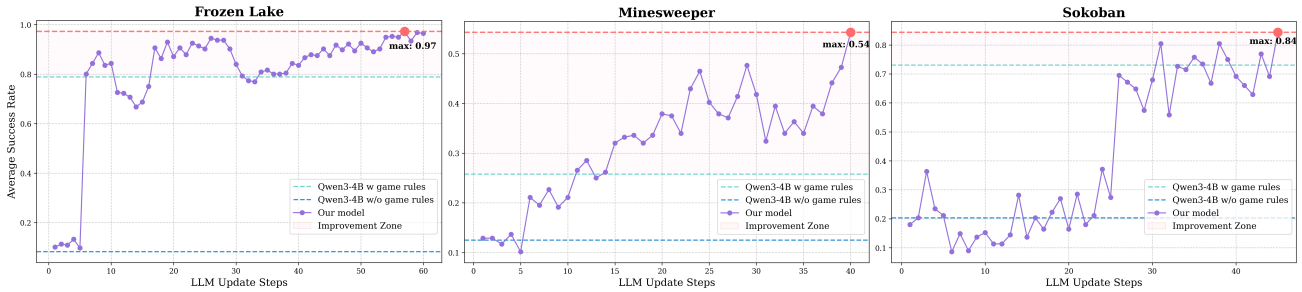


Figure 3: Learning curves of our agent on FrozenLake (left), Minesweeper (center), and Sokoban (right). The plots show the average success rate (y-axis) plotted against the number of LLM update steps (x-axis). Starting without any explicit rules, the agent's consistent improvement across these diverse tasks showcases the effectiveness of its autonomous rule discovery and policy learning.

4. Experiments

4.1. Game Environments

We evaluate our method on three classic grid-world environments: Minesweeper, Frozen Lake and Sokoban. These environments serve as common benchmarks for tasks with sparse rewards in reinforcement learning. Detailed descriptions of the environments are provided in Appendix A.

In our experimental setup, all three environments are configured with a sparse reward signal. The agent receives a reward only at the conclusion of the game, receiving +1 for successfully completing the objective and 0 otherwise. Crucially, we DO NOT provide the agent with any explicit game rules. It must learn the dynamics of each environment solely through interaction, with its knowledge limited to the set of available actions. This setup, combining sparse rewards with unknown rules, presents a significant reasoning and planning challenge.

4.2. Implementation Details

We conducted experiments using rLLM [27], backed by verl [19]. We use the Qwen3-4B-Instruct [15] model to interact with the environments. We evaluate the performance over 32 randomly sampled seeds for each game. For each of these seeds, we conduct 8 independent trials, and report the average success rate over the total 256 playthroughs per game. The rule update frequency is set to once every 5 episodes. We use GRPO [2, 18] for LLM post-training and set the maximum response length to 8,192 tokens to encourage the model to think, reason, and plan. The outcome reward is used for optimizing the LLM.

4.3. Results

Figure 3 illustrates our CEL agent's learning performance across the three environments. We compare CEL against two zero-shot baselines operating with and without the ground-truth game rules, respectively. Despite starting with no explicit game rules, the agent demonstrates a clear and positive learning trend in all tasks, validating the effectiveness of our interaction-reflection cycle. In the logical puzzle of Minesweeper, the agent exhibits steady improvement, with its success rate progressively climbing to a peak of 54%. Notably, this surpasses the 26% success rate of the baseline agent that was explicitly

provided with the ground-truth game rules, suggesting that our method of autonomous rule discovery and strategy refinement leads to a more effective policy. A different learning dynamic emerged in the complex planning puzzle of Sokoban, where the agent's performance showed a distinct "breakthrough" pattern, increasing sharply to an 84% success rate after an initial period of exploration. This highlights its ability to uncover critical insights for solving multi-step problems. The agent's efficiency was most apparent in the Frozen Lake navigation task, where it learned with remarkable speed to achieve a near-perfect success rate of 97% within the first 10 episodes. Collectively, these results showcase the general applicability and effectiveness of our approach, as it successfully masters diverse tasks ranging from logical deduction to long-horizon planning by autonomously discovering environmental rules and iteratively refining its own strategy from raw interaction.

4.4. Ablation Study

We conducted an ablation study in the Minesweeper environment to test the necessity the component of the iterative Induction of Environmental Dynamics, with results shown in Figure 4. The baseline agent, operating without the Rule Induction mechanism ("w/o Rules"), exhibits a largely flat learning curve, with its success rate stagnating at a low level. This confirms that the ability to infer and utilize a model of the environment's dynamics is fundamental to achieving competence. A second variant, which performs Rule Induction only once and then uses a static rule set ("Rules induced once"), shows initial improvement but quickly stagnates and its performance degrades, suggesting its initial rules were incomplete or inaccurate. In stark contrast, our full CEL agent, which engages in the post-episode reflection and refinement phase to continuously update its rule set $\mathcal{G}_k$, shows a robust and sustained learning trajectory, significantly outperforming both ablated versions. This comparison unequivocally demonstrates that the iterative refinement of the agent's world model is a critical component of our architecture.

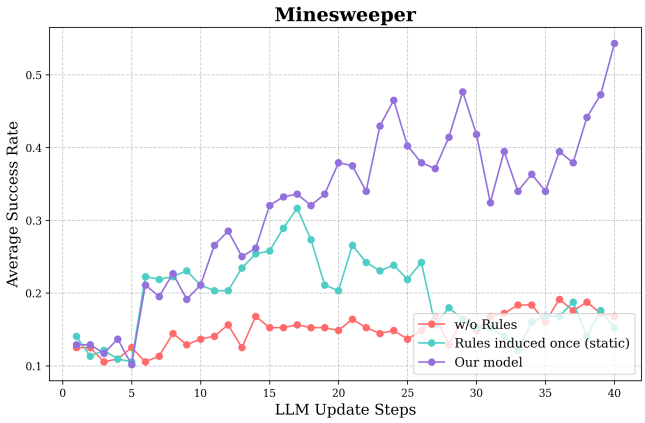


Figure 4: Ablation study illustrating the critical role of iterative **Induction of Environmental Dynamics**. The full model (blue) significantly outperforms variants with static rules (teal) or no rules (red), demonstrating that continuous refinement of the learned rulebook $\mathcal{G}_k$ is essential for sustained performance improvement.

4.5. Case Study

4.5.1. In-Episode Decision-Making

Figure 5 provides a qualitative snapshot of the agent's In-Episode Decision-Making process. The examples showcase how the agent performs a one-step lookahead search, a process relying on the synergy between its core cognitive components. First, the agent employs its Language-based Value Function (LVF) to produce a holistic, linguistic assessment of the current state's potential, $\hat{v}(s_t)$. In the Minesweeper example, it correctly identifies the state as having "high strategic value". Next, for each viable action, the agent utilizes its Language-based World Model (LWM), conditioned on its learned rules $\mathcal{G}_k$, to simulate

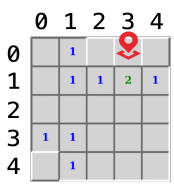
State	Prediction of the Value of Current States	Prediction of Immediate Consequences	
	The current state has a high strategic value because the only safe unrevealed cell is (0,3). Revealing it will complete the safe zones and leave only the mines unrevealed — which is acceptable in the game (as long 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 5: The agent's In-Episode Decision-Making process. At each step, the agent uses its Language-based Value Function (LVF) to assess the current state's value (middle column). It then employs its Language-based World Model (LWM) to predict the consequences of each action (right column). The agent selects the action leading to the outcome with the highest predict value.

the immediate future, predicting the next state $\hat{s}_{t+1}$ and reward $\hat{r}_{t+1}$. It accurately forecasts that the "(0, 3)" action in Minesweeper solves the puzzle. By comparing the predicted outcomes, the agent selects the action leading to the most favorable consequence, highlighting how its explicit, language-based reasoning drives intelligent planning.

4.5.2. Autonomous Rule Discovery

Figure 6 presents an example of the agent's learned rulebook $\mathcal{G}_k$ for Minesweeper, a direct output of the Induction of Environmental Dynamics process. Synthesized from its interaction trajectory and starting from a *tabula rasa* rule, the generated rules are remarkably comprehensive and accurate, covering everything from *Symbol Meanings* to the ultimate *Game Objective*. As defined in our method, this explicit, human-readable rule set $\mathcal{G}_k$ is the critical information that grounds the agent's cognitive functions. It provides the foundation for the Language-based World Model to predict future states and for the Language-based Value Function to estimate state values, thereby enabling all subsequent planning.

4.5.3. Emergent Strategy and Playbook Generation

In parallel with rule induction, our agent constructs a strategic playbook Π_k, via the Strategy and Playbook Summarization process, synthesized in Figure 7. As defined, the LLM analyzes an episode's trajectory τ_{k-1} and outcome Z_{k-1} to distill experiences into actionable advice. The emergent knowledge exhibits a sophisticated hierarchy, from tactical *Methods* like Constraint Propagation to high-level *Principles* like Safe Exploration. The discovery of these expert-level heuristics from raw interaction highlights the agent's capacity for strategic abstraction. This explicit playbook Π_k is then used alongside the rule set $\mathcal{G}_k$ in the next episode to condition the Language-based Value Function, enabling more nuanced, strategically-aware judgments and forming a direct feedback loop from experience to adaptation.

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).
...	...
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 6: An illustrative excerpt of the agent's learned rulebook $\mathcal{G}_k$ for Minesweeper, generated via the Induction of Environmental Dynamics process. Starting from no prior knowledge, the agent synthesizes a comprehensive and accurate set of rules from its interaction trajectory. Please refer to Figure 14 in the Appendix for the complete rulebook.

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.

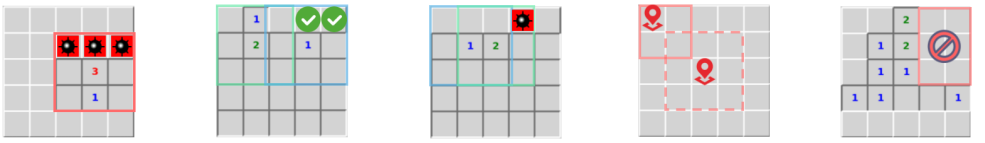


Figure 7: A synthesis of the strategic playbook Π_k for Minesweeper, generated via Strategy and Playbook Summarization. The agent distills both tactical *Methods* and high-level *Principles* from its gameplay experience. This explicit playbook is used to condition the agent's value judgments, enabling more strategically sophisticated decision-making.

4.6. Generalization

To validate that CEL agent learns by understanding rather than memorization, we tested its generalization capabilities in two settings, summarized in Table 1.

Table 1: The CEL agent's generalization performance across intra-game (unseen layouts) and inter-game (new game environments) settings. The results demonstrate the agent's strong generalization, showcasing both robust performance on unseen layouts (intra-game) and successful zero-shot transfer to novel environments (inter-game). Values in (.) show gain over the Zero-shot baseline, while those in [.] show change relative to the corresponding in-domain performance.

Trained On	Tested On (Inter-Game)		Intra-Game (Unseen layouts)
	Minesweeper	FrozenLake	
Zero-shot w/ Rule	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]

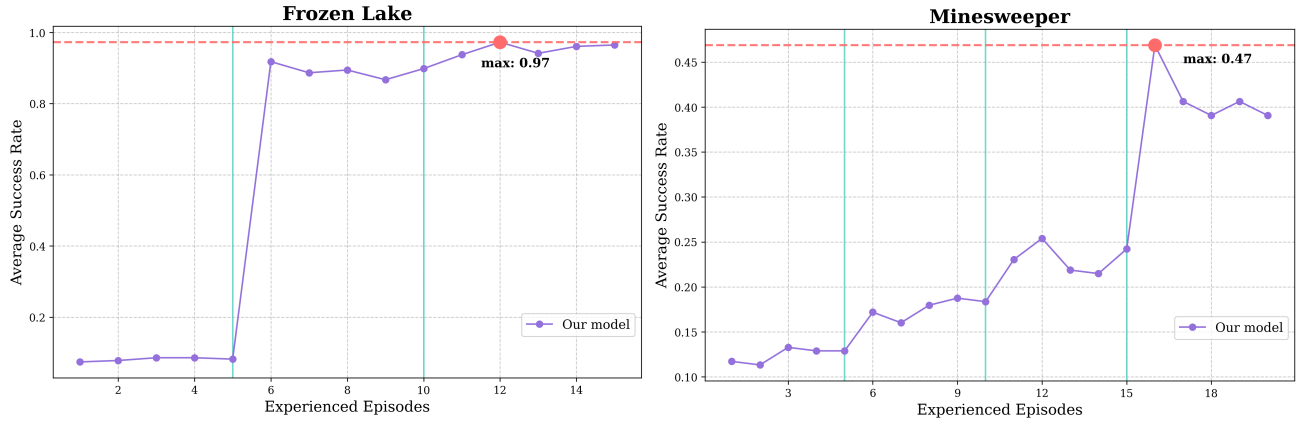


Figure 8: Inter-game generalization study showcasing adaptation to novel environments without model retraining. The plots show the agent's cross-game performance: a Minesweeper-trained agent on Frozen Lake (left) and a FrozenLake-trained agent on Minesweeper (right). In both evaluations, the core model weights remain frozen. The agent's adaptation relies solely on the iterative refinement of its explicit rulebook and strategic playbook every 5 episodes (indicated by cyan lines).

First, for **intra-game generalization**, we evaluated the agent on 32 new seeds that were entirely unseen layouts from those used during training. The agent maintained a high level of performance on these unseen 256 instances, confirming that it learns the game's fundamental principles rather than overfitting to the specific training levels. This highlights a fundamental paradigm difference from conventional reinforcement learning, which is notoriously hard to generalize to unseen domains. Instead of overfitting to learned patterns, the CEL agent's success on new layouts stems from its ability to apply an understanding of the game's rules to reason and plan effectively.

Furthermore, in the more challenging **inter-game generalization** setting, detailed in Figure 8, we tested a model trained on one game in the environment of another. Specifically, a Minesweeper-trained agent tested on Frozen Lake (left) and a FrozenLake-trained agent on Minesweeper (right) both show robust learning curves, despite their core model weights being frozen. This success indicates that the agent transfers not its knowledge of game-specific rules, but rather its fundamental ability to learn by reasoning and planning when faced and interact with a novel environment. The CEL agent thus demonstrates a sophisticated ability to generalize not the concrete dynamics of a game, but the abstract wisdom of how to *reason, plan* and then *act*.

5. Conclusions

In this work, we introduced *Cogito, ergo ludo* (CEL), a novel agent architecture that learns by explicitly reasoning about its environment. Through a unique two-phase cycle of in-episode planning and post-episode reflection, CEL autonomously constructs a human-readable world model and strategic playbook from raw interaction, starting from a tabula rasa state. Our results across several environments demonstrate that this "learning by thinking" approach allows the agent to master complex tasks while creating a transparent and auditable decision-making process. CEL marks a significant departure from opaque, brute-force learning paradigms. It validates language-based reasoning as a powerful foundation for building agents that are not only capable but also interpretable and trustworthy, opening compelling pathways toward hybrid systems that fuse CEL's explicit understanding with traditional architectural efficiency.

References

[1] 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.

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

[3] 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*, pages 1407–1416. PMLR, 2018.

[4] 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.

[5] 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.

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

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

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

[9] 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.

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

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

[12] 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.

[13] 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.

[14] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[15] Team Qwen. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

[17] 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.

[18] 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.

[19] 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.

[20] David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 1, 2025.

[21] 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.

[22] 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.

[23] 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.

[24] Rich Sutton. The Bitter Lesson, 2019. URL http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

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

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

[27] 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.

[28] 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.

[29] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[30] 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.

This appendix extends upon the main paper by providing additional details on our experimental setup, ablation studies, qualitative results, and implementation specifics.

- **Section A:** Provides descriptions of the game environments used in our experiments.
- **Section B:** Presents an analysis on the "action-only" model.
- **Section C:** Shows the results of an additional experiment on Minesweeper with an expanded training set of 128 seeds, highlighting the scalability of our agent.
- **Section D:** Contains the full prompt templates used for the agent's in-episode decision-making and post-episode reflection.
- **Section E:** Provides concrete, qualitative examples of the agent's learned knowledge, including decision-making traces, environmental rules, and strategic playbooks for various games.

A. Details of Environments

All game environments used in our experiments are from the TextArena [6]. Below are the descriptions for the specific environments and configurations used in this work.

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 the location of the mines. In our experiments, the game is configured on a 5×5 grid with 3 randomly placed mines.

Frozen Lake is a canonical grid navigation problem on a 6×6 grid where an agent must travel from a start tile to a goal tile, avoiding 6 randomly placed holes. In our deterministic setting, each action moves the agent exactly one cell in the chosen direction, removing the stochastic "slippery" nature often associated with this environment. This modification allows for a direct assessment of the agent's planning and rule-induction capabilities without the confounding factor of environmental randomness.

Sokoban is a classic puzzle game where the player must push all boxes to designated goal locations. The player can only push one box at a time and cannot pull boxes. This simple constraint creates a complex search space and necessitates careful, long-horizon planning to avoid irreversible states, such as trapping a box in a corner. The version used in our study is played on a 6×6 grid with a single box.

B. Analysis on Rollout Outcomes for Action-only Model

To understand the necessity of cognitive components and chain-of-thought reasoning, we ablated it with a simplified "Action-only" agent that directly outputs actions. When attempting to train this agent with GRPO, we observed a consistent training failure. The reason lies in the extreme polarization of rollout outcomes, as shown in Figure 9. GRPO requires batches with mixed results (i.e., "partially successful") to derive a learning signal. However, across all tested sampling sizes (8 to 64), the outcomes for the Action-only agent were always binary: for any given seed, all rollouts in a batch either succeeded or failed. As a result, the number of partially successful rollouts, i.e., the sole source of a viable training signal, was consistently zero. This lack of comparative data within batches starves the GRPO algorithm of a gradient, leading to a breakdown in training and highlighting the critical role of nuanced reasoning traces in enabling effective optimization.

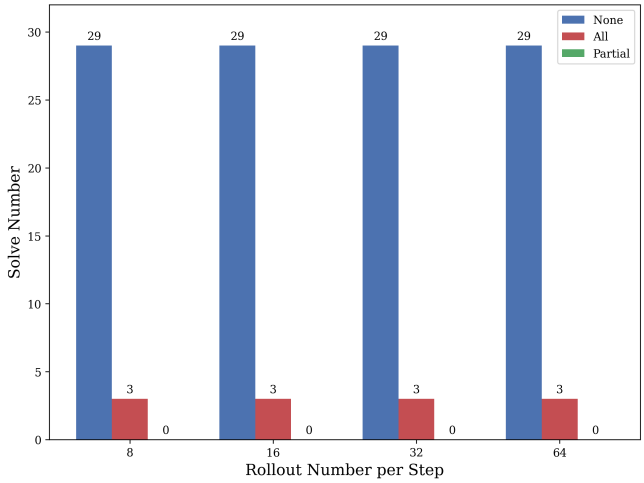


Figure 9: Distribution of rollout outcomes for the Action-only model. Across all group sampling sizes (8 to 64), outcomes are polarized into "All" (all succeed) or "None" (all fail). The number of "Partial" rollouts, which are required for GRPO to learn, is consistently zero, causing training failure.

C. Performance with an Expanded Training Set

To further evaluate the scalability and generalization capabilities of our CEL agent, we conducted an additional experiment on the Minesweeper environment. We expanded the set of training layouts by increasing the number of unique seeds from 32 (used in the main experiments) to 128.

The results are presented in Figure 10. The agent demonstrates a notable improvement in performance, with its peak success rate climbing from 54% (as reported in Figure 3) to a new maximum of 62%. This finding suggests that exposure to a more diverse set of game scenarios directly enhances the agent's core reasoning and planning capabilities. This confirms that the agent is developing a robust, generalizable problem-solving model for the game, rather than overfitting to a limited number of specific layouts.

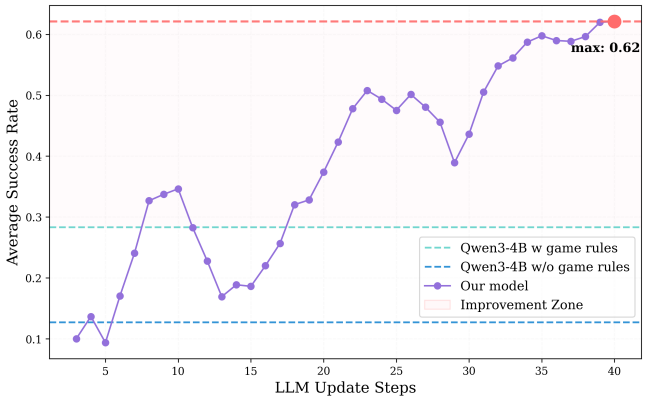


Figure 10: Learning curve for the CEL agent on Minesweeper when trained on an expanded set of 128 unique seeds. The agent achieves a new peak success rate of 62%, surpassing the performance observed with 32 seeds.

D. Prompt Templates

In this section, we present the core prompt templates used by the CEL agent. Figure 11 shows the template for in-episode decision-making, and Figure 12 shows the template for post-episode reflection.

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 11: The prompt template for in-episode decision-making (Phase 1). It instructs the LLM to evaluate the current state, assess their strategic value (LVF) and predict action outcomes (LWM).

E. Additional Results

To illustrate the explicit and interpretable knowledge base generated by our CEL agent, we provide concrete examples of Decision-Making processes (Figure 13), learned environmental rules (Figure 14, Figure 15, Figure 16) and strategic playbooks (Figure 17, Figure 18) for the Minesweeper, FrozenLake and Sokoban environment.

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 12: The prompt template for the Rule Induction and Playbook Summarization process (Phase 2). It guides the LLM to analyze a completed episode's trajectory and refine its explicit model of the environment's dynamics.

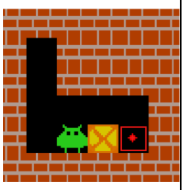
State	Prediction of the Value of Current States	Prediction of Immediate Consequences	
	High value. The player is in a direct line of action to push a box into the goal. The box is one step away from the goal, and the direction of push is clear. This is a high-probability win scenario with only one valid 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 13: An example of a Decision-Making process for Sokoban environment.

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 14: An example of a learned environmental rule for Minesweeper environment.

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 15: An example of a learned environmental rule for FrozenLake environment.

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 16: An example of a learned environmental rule for Sokoban environment.

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 17: An example of a learned strategic guideline from the agent's playbook for FrozenLake environment.

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 18: An example of a learned strategic guideline from the agent's playbook for Sokoban environment.