Improving Sequential Decision-Making in LLM Agents via Experience Memory

Abstract

Large language models have improved substantially on single-shot reasoning tasks, but their performance in sequential decision-making is less well understood. We study this on fully-observable two-player zero-sum games, which provide ground-truth evaluation: outcomes are determined by the rules, and optimality of individual moves can be computed or approximated, without relying on a judge model. Across model tiers, LLMs play sub-optimally in simple games such as tic-tac-toe or Connect Four, and lose to MCTS opponents. Obfuscations that preserve the game tree but rewrite its surface form leave performance largely unchanged, indicating the gap is not fully explained by recall of memorized strategies. Motivated by this performance gap, we introduce an agentic framework enhanced with an experience memory designed for the sequential setting and addressing common challenges of sequential decision-making such as credit assignment. We show that experience memory, post-game reflection, natural language state representation, and rule extraction yield consistent improvements on these games without modifying the model weights.

1 Introduction

Large language models (LLMs) now solve impressive reasoning problems, from competition mathematics to multi-file coding tasks. Yet a growing body of evidence, most visibly the recent study on the limits of "thinking" models [Shojaee *et al.*, 2025], shows that this competence does not transfer cleanly to simple sequential decision-making puzzles. Even on tasks that humans master with little practice, frontier models stumble once decisions must be chained, the environment evolves between steps, an adversary pushes back, and rewards arrive only at the end. Closing this gap is a prerequisite for deploying LLMs as autonomous agents, but progress has been hard to measure, because most sequential benchmarks rely on LLM judges or proxy metrics that conflate reasoning quality with surface plausibility.

We study this gap in a setting that admits ground-truth evaluation: perfect-information two-player zero-sum games. Outcomes are determined by the rules, the optimality of each individual move can be computed exactly by classical solvers for games of modest size, and the games we consider, tic-tac-toe, Nim, and Connect Four, are simple enough that a casual human player can master them. Building on GTBench [Duan *et al.*, 2024], we evaluate state-of-the-art models from OpenAI, Google, and the open-source community across three capability tiers. The picture is consistent and discouraging. Efficient and Intermediate models miss five to ten optimal moves per hundred on tic-tac-toe and a non-trivial fraction on Nim, and the Frontier tier, GPT-5.4, Gemini 3.1 Pro, and GLM-5, lose to a moderate MCTS opponent on Connect Four in essentially every game, typically collapsing within ten moves of a forty-move game. To probe how robust this behavior is, we evaluate the same models under a suite of game-rule obfuscations that preserve the underlying game tree but rewrite its surface form. Performance is largely invariant across formulations, and reasoning traces frequently identify the underlying game by name even when its rules have been disguised. The obfuscations therefore do not cleanly separate reasoning from recall, models can still draw on memorized strategies once they recognize the game, but they do show that the failures are stable properties of how each model engages with these positions, not artifacts of a particular representation. Whatever the source, what the model produces at decision time is insufficient on its own.

A natural response would be to fine-tune the models on game play, but updating frontier-model weights is costly, often inaccessible, and risks regressing the capabilities the agent depends on for parsing rules and planning over them. We therefore ask whether the gap can be closed at the agent level, by supplying the missing decision-time signal externally and leaving the model untouched. Prior work on case-based reasoning and self-reflection, Memento [Zhou *et al.*, 2025], ExpeL [Zhao *et al.*, 2024], and Reflexion [Shinn *et al.*, 2023], has shown that experience memory and generating reflective feedback can yield consistent gains on static question-answering and tool-use tasks without weight updates. These methods, however, were not designed for environments in which the state evolves adversarially and the reward arrives only at the end of an episode. A direct port to game play is essentially non-functional in our experiments.

We introduce RECAP, a reflective case-based reasoning framework that adapts experience memory to sequential play.

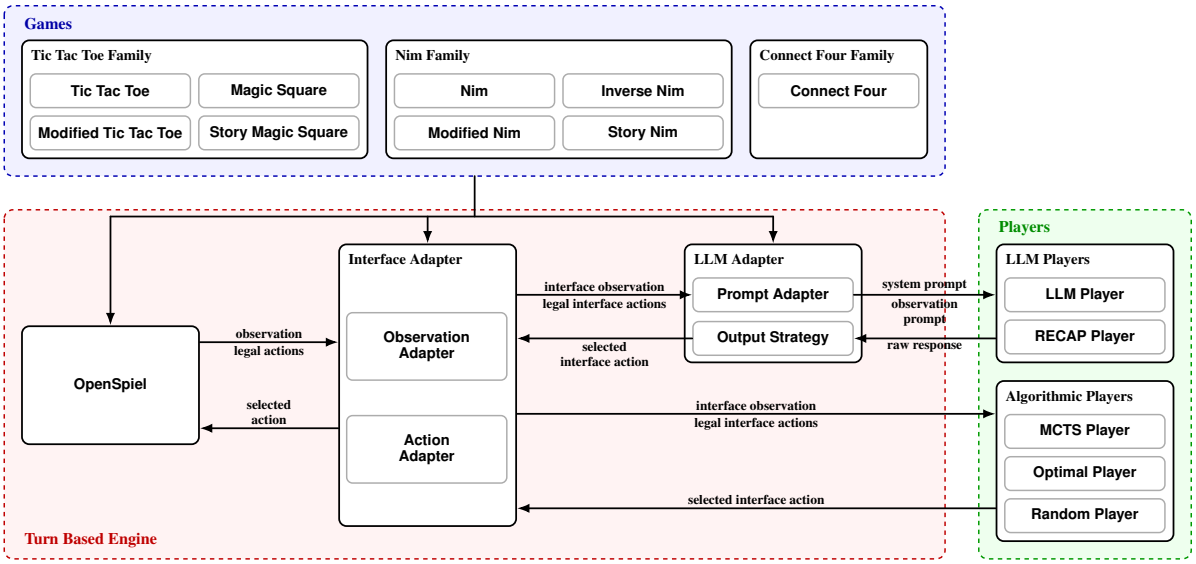


Figure 1: A simplified schema of OmniPlay highlighting the modularity and extensibility of the framework to add new games and players.

RECAP keeps the LLM as the primary reasoner and adds three components targeted at the structural challenges of the setting: a self-reflector that performs per-move credit assignment after each completed trajectory and emits fine-grained annotations augmenting the terminal reward; a state descriptor that produces a compact natural-language summary of the current position so that retrieval operates on the kind of input the embedding model was trained for; and a rule extractor that periodically distills accumulated cases into a small set of natural-language strategic rules. RECAP requires no weight updates and learns purely from played episodes substantially improving over both the base LLM and a case-based-reasoning baseline on tic-tac-toe with GPT-5 nano against the Optimal opponent.

Our contributions are:

- An empirical characterization of the sequential-reasoning gap in current LLMs on simple perfect-information two-player zero-sum games, showing that the gap persists across model tiers and providers and is largely invariant to surface representation.
- RECAP, an agentic framework that adapts case-based reasoning to sequential decision-making through per-move reflection, natural-language state descriptors, and rule extraction, and that improves performance over a strong baseline without modifying the model weights.

2 Related Work

A growing body of work has examined how LLMs perform on reasoning tasks that require more than a single output step. Chain-of-Thought prompting [Wei *et al.*, 2022] elicits intermediate reasoning steps and has become standard practice on multi-step problems, with extensions such as self-consistency [Wang *et al.*, 2023] improving robustness through majority voting over independent reasoning chains. Tree of Thoughts [Yao *et al.*, 2023a] goes further by reframing reasoning as a search problem, structuring intermediate steps into a tree and self-evaluating them to identify dead ends and backtrack. Although closer to the explicit search used in sequential decision-making, it operates over the model's intermediate thoughts rather than over an external state that the agent can act on.

A separate line of research targets environments with external state directly. ReAct [Yao *et al.*, 2023b] interleaves reasoning and action steps in interactive environments, laying the foundations for agentic systems. RAP [Hao *et al.*, 2023] prompts the LLM to act as both a world model and a value function, and plans over its own predictions using Monte Carlo Tree Search. Code World Models [Lehrach *et al.*, 2025] have the LLM synthesize executable code that represents the environment, then plan over the resulting world model. These methods address reasoning limitations by wrapping the LLM in external algorithms, with the model serving as one component of a larger scaffold.

A complementary approach is to keep the LLM as the primary reasoner and provide it with structured context drawn from past experience. Memento [Zhou *et al.*, 2025] introduces a case-based reasoning architecture in which a planner-executor pair consults a growing Case Bank of past task-solving trajectories, retrieving relevant cases at decision time. ExpeL [Zhao *et al.*, 2024] extracts natural-language insights from successful and failed trajectories and supplies them as additional context for new tasks. Reflexion [Shinn *et al.*, 2023] prompts the agent to reflect on its own failures and uses the resulting reflections as guidance on subsequent attempts at the same task. These methods establish that experience memory and self-reflection can yield consistent gains without modifying model weights, but they were developed primarily for static question-answering and tool-use tasks, where the environment does not respond adversarially and trajectories are short enough that storing them as a single unit, labeled by the final outcome, gives a useful retrieval signal. RECAP adapts this family of methods to sequential play, where tra-

jectories are longer, the reward is a single binary win/loss arriving only at the end, and the same trajectory routinely mixes strong moves with the one mistake that decided the game, so a per-trajectory label is too coarse to drive retrieval or learning.

A third line of work focuses on the infrastructure for evaluating LLMs on games. OpenSpiel [Lanctot *et al.*, 2019] provides a comprehensive library of game implementations and reference algorithms, and is widely used as the standard engine for game-theoretic experiments. Building on OpenSpiel, GTBench [Duan *et al.*, 2024] introduces a benchmark suite for evaluating LLMs on a range of perfect- and imperfect-information, deterministic and stochastic games, and reports substantial gaps to optimal play, including on tic-tac-toe, Nim, and Connect Four. OmniPlay builds on GTBench and extends it into a modular evaluation framework that supports arbitrary players, games, and obfuscations through a small set of adapter interfaces, and serves as the platform on which we evaluate RECAP.

3 The Sequential Reasoning Gap

In this section we diagnose the sequential reasoning gap by evaluating LLMs on simple, perfect-information two-player zero-sum games. As argued in Section 1, such games provide a well-controlled setting for this purpose: the rules are strict, individual actions and final outcomes enable unambiguous ground-truth evaluation, and for games of modest size the optimal policy is computable exactly. We first show that current state-of-the-art LLMs remain unreliable even in simple domains, then probe whether this unreliability is better explained by recall of memorized strategies or by flawed reasoning about the current state. We show that the gap to optimal play persists across models and opponents and is largely invariant to representation, motivating the method introduced in Section 4.

3.1 OmniPlay

To evaluate a broad spectrum of players, ranging from random and algorithmic baselines to various LLM-based players, on arbitrary games, we developed the evaluation framework OmniPlay[1], shown in Figure 1.

OmniPlay is built on top of OpenSpiel [Lanctot *et al.*, 2019], a widely used library of game implementations and reference algorithms. It extends OpenSpiel with two components, the **Interface Adapter**, which translates observations and actions into a unified representation suitable for all kinds of players, and the **LLM Adapter**, which transforms this representation into a text-based format suitable for LLMs. New games and players are integrated by implementing methods for these adapters, regardless of internal complexity. OmniPlay then handles correct game execution, player orchestration, result aggregation and statistical analysis.

3.2 Evaluation Setting

We consider three representative perfect-information two-player zero-sum games, each posing different challenges. **Tic-tac-toe** has a small game tree and short-horizon planning, with a draw as the optimal outcome, and therefore tests the player's ability to avoid losing rather than to force a win. **Nim** also has a small game tree, but requires a less obvious strategy based on the Nim-sum of the piles, and tests the player's ability to force a win from the starting position. **Connect Four** has a much larger game tree and requires long-horizon planning together with anticipating and creating multiple simultaneous threats.

To measure the game-playing ability of the players we use three opponents of increasing strength. The **Random** opponent chooses actions uniformly at random. The **Monte-Carlo Tree Search (MCTS)** opponent is a strong but exploitable baseline. Since MCTS estimates state values from random rollouts, a bounded simulation budget n leaves a non-trivial probability of suboptimal play. We use $n = 1000$ for tic-tac-toe and Nim and $n = 5000$ for Connect Four, large enough for strong play but small enough to leave occasional mistakes. Other hyperparameters are set to the OpenSpiel defaults. The **Optimal** opponent runs the *Minimax* algorithm with uniformly random tie-breaking.

We evaluate state-of-the-art LLMs from OpenAI [Singh *et al.*, 2026] and Google [Google, 2025] together with selected open-source models [Yang *et al.*, 2025; GLM-5-Team *et al.*, 2026], grouped into three tiers (see Table 1) by expected capability determined by public benchmarks [Analysis, 2026] rather than by parameter count. Unless stated otherwise, the models are queried with reasoning effort set to *high*, if such a setting is exposed by the provider, and all other parameters are left at default values.

Group	Models
Efficient	GPT-5 nano, Gemini 3.1 Flash Lite, GPT-OSS 120B
Intermediate	GPT-5 mini, Gemini 3 Flash, Qwen 3.5 122B
Frontier	GPT-5.4, Gemini 3.1 Pro, GLM-5

Table 1: LLMs grouped by expected capabilities into tiers.

Under optimal play, tic-tac-toe is a draw for the first player, whereas the selected Nim initial position $\{1, 3, 5, 7\}$ and standard Connect Four are first-player wins. For each player-opponent pair, we run 100 games and switch sides after half of the games to control for the first-player advantage. For tic-tac-toe and Nim, we report per-move optimality rates, where a move is counted as optimal if it could have been chosen by Minimax. States in which all legal moves are optimal are excluded because they provide no discriminative signal. For Connect Four, where exact optimality is not easily evaluated, we instead report win rate and average number of player moves in the game.

3.3 Failures in Simple Domains

We first evaluate the LLMs under the standard game rules. We found the Frontier tier to play tic-tac-toe and Nim optimally in this setting, so we restrict the analysis of these two games to the Efficient and Intermediate tiers. The Frontier tier is evaluated below on Connect Four, where optimal play is substantially more challenging.

Table 2 shows that no model from the Efficient or Intermediate tier reaches optimal play on tic-tac-toe, even with high

[1]The code for OmniPlay is publicly available at [anonymized]

Model	tic-tac-toe	Nim
GPT-5 nano	.880 [.828, .918]	.955 [.904, .979]
Gemini 3.1 FL	**.956** [.918, .977]	**1.000** [.976, 1.000]
GPT-OSS	.901 [.853, .935]	.980 [.943, .993]
GPT-5 mini	.922 [.878, .952]	.993 [.959, .999]
Gemini 3 F	**.995** [.974, .999]	**1.000** [.974, 1.000]
Qwen 3.5	.949 [.909, .972]	**1.000** [.974, 1.000]

Table 2: Per-move optimality rates with 95% Wilson score confidence intervals for Efficient- and Intermediate-tier LLMs on tic-tac-toe and Nim against the Optimal opponent. Each model plays 100 games, switching sides halfway through.

reasoning effort, and several models occasionally fail to produce a legal action. Apart from Gemini 3 Flash, which makes roughly one mistake per 200 moves, the remaining models make five to ten mistakes per 100 moves, a substantial error rate given the small game tree. On Nim, the picture is mixed: several models reach optimal play, but the smaller models still miss a non-trivial fraction of moves.

The situation is significantly worse for the Frontier tier on Connect Four. Table 3 reports win rates and the average number of moves made by the LLM player against the Random and MCTS opponents. All three models defeat the Random opponent in every game, indicating that they can follow the rules and exploit unstructured play. However, given the low probability of strong moves under random play, this is only a weak indicator of strategic competence.

Model	Random		MCTS	
	Win rate	Moves	Win rate	Moves
GPT-5.4	**1.00** [0.72, 1.00]	4.90 [4.04, 5.76]	0.00 [0.00, 0.28]	10.10 [6.89, 13.31]
Gemini 3.1 Pro	**1.00** [0.72, 1.00]	5.40 [4.43, 6.37]	**0.10** [0.02, 0.4]	10.70 [6.99, 14.41]
GLM-5	**1.00** [0.72, 1.00]	6.10 [5.01, 7.19]	0.00 [0.00, 0.28]	7.60 [6.12, 9.08]

Table 3: Win rate and average number of moves made by the LLM player on Connect Four against the Random and MCTS opponents. Win rates are reported with 95% Wilson score confidence intervals, while the number of LLM moves is reported with 95% Student's t confidence intervals. Each model plays 10 games against each opponent, switching sides halfway through.

Against the non-perfect MCTS opponent, only Gemini 3.1 Pro managed a single win out of ten games, exploiting an opponent mistake, while the other two lost every game. Under bilateral perfect play, Connect Four ends with a first-player win on move 41, so each player makes at least 20 moves. Against MCTS, however, the Frontier models make only 9.5 moves on average before losing, indicating failure in the opening phase rather than in tactically complex middlegames.

3.4 Obfuscations

We have seen that models fail even on simple games such as tic-tac-toe and Nim. Both games are well known and likely appear frequently in pretraining data, including examples of good and bad play from many positions. A natural alternative explanation is therefore that the observed errors reflect imperfect recall of familiar positions or strategies, rather than reasoning about the current state.

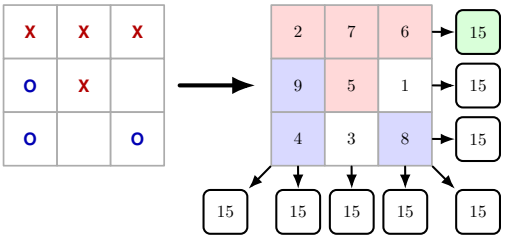


Figure 2: A Magic Square obfuscation of tic-tac-toe. The original game board is mapped to a Magic Square grid, where each cell is assigned a number from 1 to 9 such that the sum of each row, column and diagonal is 15. The players choose numbers from a set of numbers, the first one to collect three numbers summing to 15 wins the game.

To probe the role of memorized strategies, we introduce alternative rule formulations that preserve the underlying game tree while changing its surface representation. Each obfuscation is structurally identical to the original game, but the state observations and available actions, both expressed in natural language, are different.

We focus here on the tic-tac-toe obfuscations, which provide a compact representative case, Nim is evaluated with analogous surface transformations and presented in Supplementary B. Since the exact obfuscated formulations are unlikely to appear in pretraining data, a model relying primarily on memorized surface formulations or positions tied to the original representation should perform noticeably worse on them.

Hidden tic-tac-toe (Hid) uses the same rules as tic-tac-toe except that the markers `X` and `O` are replaced with `B` and `W` and the game name is not mentioned. **Magic Square** (M.Sq) is played on a set of numbers $\{1, \ldots, 9\}$. Each cell of the original board is assigned one number from this set so that every row, column, and diagonal sums to 15. Players alternate taking numbers, and the first player to hold any three numbers summing to 15 wins. An example configuration is shown in Figure 2. **Magic Square Add** 5 (M.Sq +5) is identical to Magic Square, but every cell value is incremented by 5 to obscure the standard pattern. **Story Magic Square** (Story) encodes the M.Sq obfuscation as a story. Each player tries to cross a river of length 15 by composing a sequence of exactly three jumps. Players take turns choosing jump lengths from $\{1, \ldots, 9\}$, each jump length may be used only once, and the first player to cross the river exactly, without overshooting or undershooting, wins.

The evaluation setting mirrors that of Section 3.3. Across models, we find no consistent performance gap between the original and obfuscated formulations of tic-tac-toe. As shown

Model	TTT	Hid	M.Sq	M.Sq +5	Story
GPT-5 nano	.880 [.828, .918]	.871 [.817, .911]	.888 [.837, .925]	.785 [.718, .840]	**.844** [.786, .888]
Gemini 3.1 FL	**.956** [.918, .977]	**.950** [.910, .972]	**.991** [.966, .997]	**.991** [.966, .997]	.703 [.635, .763]
GPT-OSS	.901 [.853, .935]	.887 [.837, .924]	.901 [.853, .935]	.779 [.716, .832]	.835 [.777, .880]
GPT-5 mini	.922 [.878, .952]	.923 [.878, .952]	.967 [.934, .984]	.971 [.939, .987]	.953 [.916, .974]
Gemini 3 F	**.995** [.974, .999]	**1.00** [.983, 1.00]	**1.00** [.983, 1.00]	**.995** [.974, .999]	**1.00** [.983, 1.00]
Qwen 3.5	.949 [.909, .972]	.956 [.918, .977]	**1.00** [.982, 1.00]	.986 [.960, .995]	.721 [.656, .779]

Table 4: Per-move optimality rates with 95% Wilson score confidence intervals for Efficient- and Intermediate-tier LLMs on tic-tac-toe and its obfuscations against the Optimal opponent. Each model plays 100 games per rule formulation, switching sides halfway through.

in Table 4, the main exceptions are Gemini 3.1 Flash Lite and Qwen 3.5 122B on Story Magic Square, and GPT-OSS 120B on Magic Square Add 5, where optimality drops by a meaningful margin. Further inspection of the available reasoning-trace summaries would be necessary to fully understand the sources of confusion. Overall, these results suggest that the models are not relying solely on memorized tic-tac-toe positions, but can often adapt their play to semantically equivalent rule formulations.

Model	Hid	M.Sq	M.Sq +5	Story
GPT-5 nano	.830 [.791, .863]	.931 [.902, .951]	.424 [.377, .472]	.482 [.435, .530]
Gemini 3.1 FL	**.998** [.987, 1.00]	.922 [.893, .943]	.256 [.218, .298]	.189 [.154, .231]
GPT-OSS	.944 [.918, .963]	**.951** [.926, .968]	**.539** [.486, .591]	**.736** [.691, .776]
GPT-5 mini	.364 [.317, .414]	.588 [.538, .635]	.581 [.534, .628]	**.859** [.822, .889]
Gemini 3 F	.991 [.977, .997]	.831 [.794, .863]	.414 [.369, .460]	.756 [.714, .793]
Qwen 3.5	**1.00** [.991, 1.00]	**.996** [.984, .999]	**.633** [.587, .676]	.435 [.387, .485]

Table 5: Proportion of reasoning traces that explicitly identify the underlying game as tic-tac-toe under each obfuscation. Results are aggregated over 100 games per model-obfuscation pair ($\sim$ 400 traces per cell), and reported with 95% Wilson score confidence intervals.

To further inspect the reasoning process, we examine reasoning-trace summaries and compute the proportion that explicitly names the underlying game, tic-tac-toe in this case. As shown in Table 5, models identify the underlying game in a substantial fraction of traces across all obfuscations. This suggests that the models can often infer the hidden structure from the rule description, rather than merely matching the original surface form. At the same time, once the hidden structure is recognized, we cannot rule out the use of memorized tic-tac-toe positions or strategies.

Although optimality rates vary little across obfuscations, the number of output tokens per move increases steadily with obfuscation complexity, with token statistics reported in Supplementary C.1. This suggests that models expend additional reasoning effort to interpret the harder formulations and map them to playable states.

Overall, this section shows that current state-of-the-art LLMs remain unreliable in simple sequential decision-making domains. Models in the Efficient and Intermediate tiers fall short on tic-tac-toe and Nim, while the Frontier tier fails early on Connect Four. Performance remains largely invariant across surface-level obfuscations, and reasoning traces show that models frequently identify the obscured game from the rules alone. Together, these findings argue against a pure surface-memorization explanation and point to a decision-time reasoning bottleneck. Additional results are reported in Supplementary A, Supplementary B, and Supplementary C.

4 Reflective Case-Based Reasoning

Section 3 showed that LLMs remain unreliable in simple perfect-information games. This unreliability is largely invariant to surface-level changes in the game rules, suggesting a decision-time reasoning bottleneck rather than faulty recall of strategies observed during training. A natural way to address this bottleneck without modifying model weights is to provide the model with structured context from past experience, as Memento [Zhou *et al.*, 2025] and ExpeL [Zhao *et al.*, 2024] do for question-answering and tool-use tasks. Although these tasks may involve sequential reasoning, they differ from the sequential decision-making setting studied here. The environment does not evolve in response to the agent, and there is no adversary whose actions shape the outcome.

Adapting case-based reasoning to our setting raises challenges that these methods do not address. The unit of experience is no longer a completed task, but an individual move within an unfolding episode. The reward is observed only at the end of the episode and must be assigned across earlier moves. Retrieval must cope with structured but textually variable observations, which may be poorly aligned with standard text embedding models. We instantiate these requirements in RECAP, a reflective case-based reasoning framework that keeps the LLM as the primary reasoner while augmenting it with an experience memory designed for sequential play. We first describe the progression from a direct Memento port to a working baseline and then introduce the additional components of RECAP.

4.1 From Static Case Memory to Sequential Play

We begin with a minimal adaptation of Memento [Zhou *et al.*, 2025] to sequential game play, which we call *Original*. Memento uses an LLM meta-planner that either emits a plan or produces a final answer, and an LLM executor that solves the generated subtasks before control returns to the meta-planner. Completed task attempts are stored in case memory together with the latest plan and an outcome score produced by an LLM judge. At decision time, similar cases are retrieved to condition the solution of future tasks. To adapt this structure to games, we add a game buffer that stores each move un-

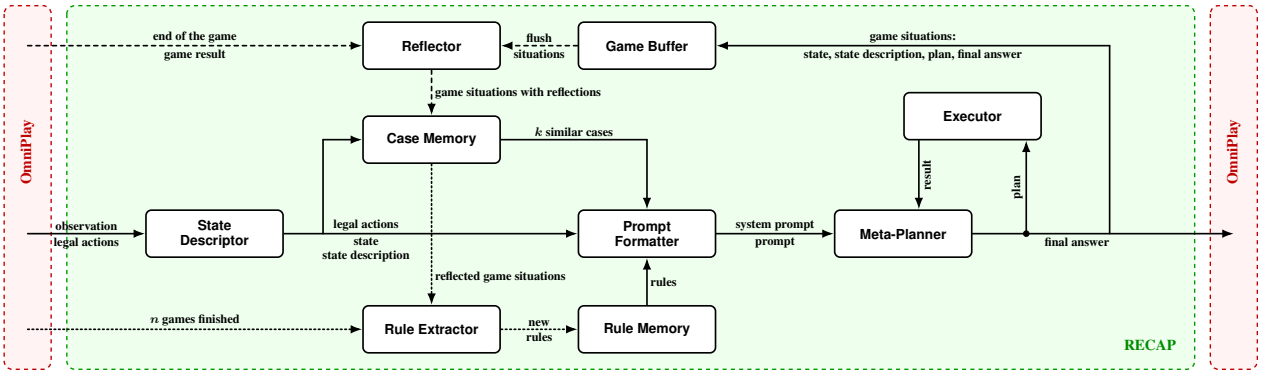


Figure 3: Schema of the RECAP architecture and it's interaction with OmniPlay.

til the episode terminates. Once the terminal outcome is observed, the same reward is assigned to all buffered moves, and the resulting cases are written to memory. Since game outcomes are determined exactly by the environment, no LLM judge is needed.

This minimal port is sufficient to make the agent interact with the environment, but it is not a reliable game-playing agent. Its prompt-level formatting and raw-response parsing are brittle in settings that require precise legal actions, and storing each move independently can flood memory with duplicate state-action pairs carrying inconsistent terminal rewards.

We therefore introduce a stronger engineered variant, denoted *Baseline*. It preserves the planner–executor loop and case memory, but improves the robustness of communication between components. The planner and executor communicate through explicit JSON schemas, replacing prompt-only formatting instructions with structured outputs and ensuring that intermediate messages are parseable. The final action is also returned inside the schema, but its value remains a free-form string so that the method does not depend on a game-specific action encoding. Memory entries are deduplicated by state-action pairs, preventing repeated identical decisions from dominating retrieval. Retrieval additionally uses stratified sampling to balance positive and negative examples.

Although these changes make *Baseline* a usable game-playing agent, they do not address two structural limitations of case-based reasoning in games. First, credit assignment remains coarse. Every move in a lost game receives negative credit, even if it was locally optimal, and every move in a won game receives positive credit, even if it was locally suboptimal. Second, retrieval remains difficult because raw game observations may be symbolic or semi-structured, making them poorly suited to embedding-based similarity search.

4.2 RECAP

RECAP, shown in Figure 3, keeps the baseline structure, but augments it with components that address the credit-assignment and retrieval issues while also adding an abstraction mechanism over accumulated play. The **self-reflector** performs trajectory-level analysis after each game and produces structured per-move reflections that are written to case memory alongside the terminal reward. These reflections capture both the local quality of a move and its contribution to the final outcome, providing a finer-grained credit signal than uniform trajectory-level reward propagation. The **state descriptor** converts structured game observations into natural-language descriptions before embedding, making retrieval more compatible with standard text embedding models. The **rule extractor** periodically consolidates accumulated experience into general natural-language strategic rules that guide the agent's reasoning in future positions.

The architecture of RECAP consists of an online decision loop that selects actions during play, and two post-game update mechanisms that expand and refine the agent's memory. On each move, the agent receives the current observation and legal actions from OmniPlay. The state descriptor converts the observation into a natural-language description, which is embedded and used as the retrieval key for case memory. Case memory returns the k most semantically similar cases, while rule memory supplies the current set of strategic rules. A prompt formatter combines the original observation, legal actions, retrieved cases, and rules into the meta-planner prompt. The meta-planner and executor then run in a loop until the meta-planner produces a final action. The selected action is passed to the environment, and the state, state description, selected action, and action intent are appended to the game buffer.

When the episode terminates, the delayed outcome is passed together with the buffered trajectory to the self-reflector. The self-reflector generates per-move reflections and writes the reflected cases to case memory. Periodically, after a fixed number of completed games, the rule extractor receives the current rules and the cases accumulated since the previous update. It then produces an updated set of strategic rules that replaces the previous rule memory.

Additional details are deferred to Supplementary D.

5 Experiments

We evaluate whether reflective case-based memory improves sequential decision-making in a setting where the base model leaves a measurable gap to optimal play. We focus on standard tic-tac-toe with GPT-5 nano at medium reasoning effort against the Optimal opponent. This setting admits exact move-level evaluation, yet Section 3 showed that Efficient

models still make frequent non-trivial mistakes on it. It therefore provides a controlled testbed for testing whether RECAP improves over both the direct Memento port, *Original*, and the engineered *Baseline*.

5.1 Evaluation Setting

Each run alternates between learning and evaluation phases. Before learning begins, the agent is evaluated for $N = 30$ games with an empty case memory. The run then proceeds for $K = 5$ epochs, each consisting of $M = 50$ learning games followed by $N = 30$ evaluation games. During learning, the agent's decisions are buffered, self-reflections are generated after each completed game, and case memory is updated. At the end of the learning phase, the rule extractor is invoked and a new rule set is generated from the previous rules and the cases accumulated during that phase. During evaluation, all memories are set to read-only, the self-reflector is disabled, and the agent is evaluated without further updates. The evaluation phase therefore measures the policy induced by the current memory snapshot. To estimate variability, each configuration is repeated for $R = 10$ independent runs.

We report draw rate as the main outcome metric, since it is the best achievable result in tic-tac-toe against the Optimal opponent. We also report output tokens per move, consisting mainly of reasoning tokens, as a measure of inference cost and agent efficiency. Error bars show 95% Student's t confidence intervals over the R independent runs.

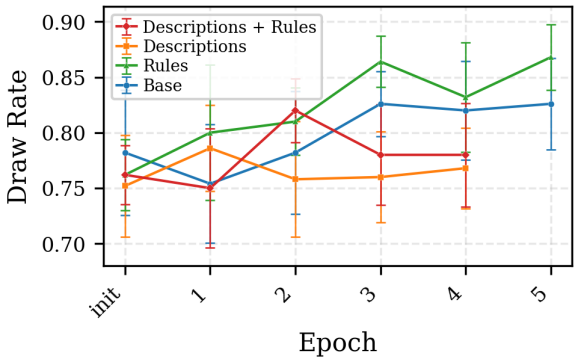


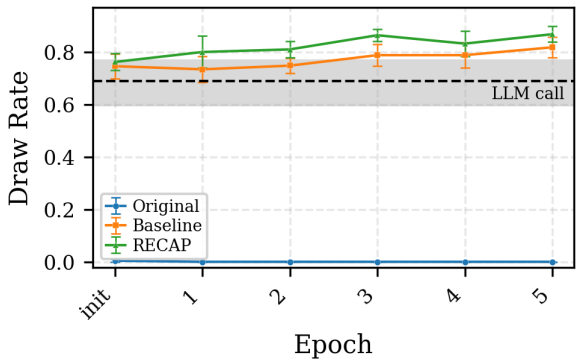
Figure 4: Ablation of individual RECAP components using GPT-5 nano with medium reasoning effort against the Optimal opponent on tic-tac-toe. Results are averaged over $R = 10$ runs, error bars show 95% Student's t confidence intervals.

5.2 Ablation of RECAP Components

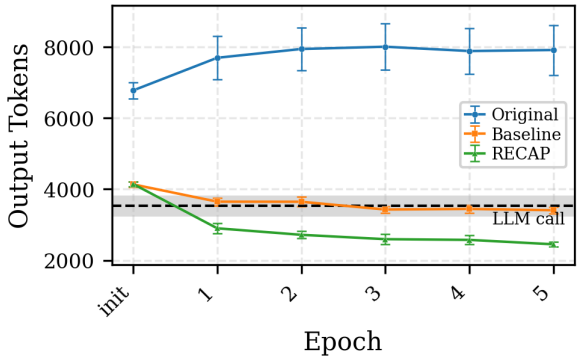
We first ablate the components introduced by RECAP. All ablated variants include the self-reflector, since per-move reflection is the core mechanism used to address delayed credit assignment. We then independently toggle the state descriptor and rule extractor, obtaining four variants: reflection only, reflection with state descriptions, reflection with rules, and reflection with both state descriptions and rules.

Figure 4 shows that reflection paired with rule extraction is the best-performing variant. The reflection-only variant reaches a final mean draw rate of 0.826, while adding rules increases it to 0.868. Although the 95% Student's t confidence intervals in Figure 4 overlap, this improvement is significant under a one-sided Welch t-test ($p = 0.039$). The same pattern appears in move-level optimality, where reflection with rules improves from 0.9586 to 0.9697 under the same test ($p = 0.027$).

In this setting, state descriptions do not improve performance and can even reduce the benefit of reflections and rules. This suggests that the standard tic-tac-toe state representation is already adequate for retrieval, or that generated descriptions introduce variability that offsets their semantic benefit.



(a) Draw rate



(b) Output tokens, including reasoning tokens, per move.

Figure 5: Comparison of the *Original* Memento port, the engineered *Baseline*, and the best *RECAP* variant on tic-tac-toe with GPT-5 nano (medium) against the Optimal opponent. The dashed line shows the single-call LLM reference from Section 3. Lines show means over 10 independent runs, with error bars denoting 95% Student's t confidence intervals. RECAP improves draw rate over Baseline while reducing output-token usage.

5.3 Improvement over Baseline and Original

We next compare the best-performing RECAP variant, reflection with rules, against the direct Memento port and the engineered Baseline. We also include the single-call GPT-5 nano player from Section 3 as a fixed reference.

Figure 5 shows that the direct Memento port, *Original*, fails to produce a usable game-playing agent. Its draw rate remains at zero across evaluation checkpoints, with most games ending in failure due to invalid moves or failed parsing. In

contrast, *Baseline* reaches a final mean draw rate of 0.818, showing that structured planner-executor communication and memory deduplication are necessary to make play viable.

The best RECAP variant improves further, reaching a final mean draw rate of 0.868. This corresponds to a gain of 5.0 percentage points over *Baseline*, which is significant under a one-sided Welch t-test ($p = 0.018$). Thus, post-game reflection and rule extraction improve performance significantly beyond the engineering changes needed to make the case-based agent reliable.

Figure 5 also reports output tokens per move. The best RECAP variant reduces token usage over training and ends below both *Baseline* and even the single-call reference. The performance gain therefore does not come from longer reasoning traces, but from more useful decision-time context.

6 Conclusion

We studied sequential decision-making in LLMs through perfect-information two-player zero-sum games, where both outcomes and individual moves can be evaluated against ground truth. Across models, we found that even simple games expose a persistent decision-time reasoning gap, and that this gap is largely invariant to surface-level rule obfuscations. We introduced RECAP, a reflective case-based reasoning framework that adapts experience memory to sequential play through structured planner-executor communication, per-move reflection, and rule extraction. On tic-tac-toe with GPT-5 nano, RECAP improves over both a direct Memento port and a strong engineered baseline, while reducing output-token usage. These results suggest that external experience memory can improve LLM agents without modifying model weights, but they also leave open how well the approach scales to larger games, longer horizons, and more complex state spaces.

References

[Analysis, 2026] Artificial Analysis. Comparison of ai models across intelligence, performance, and price, 2026.

[Duan *et al.*, 2024] Jinhao Duan, Renming Zhang, James Diffenderfer, Bhavya Kailkhura et al. Gtbench: Uncovering the strategic reasoning limitations of llms via game-theoretic evaluations. *arXiv preprint arXiv:2402.12348*, 2024.

[Gao *et al.*, 2021] Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[GLM-5-Team *et al.*, 2026] GLM-5-Team, Aohan Zeng, Xin Lv, Zhenyu Hou et al. Glm-5: from vibe coding to agentic engineering, 2026.

[Google, 2025] Google. A new era of intelligence with gemini 3, November 2025. Accessed: 2026-05-07.

[Hao *et al.*, 2023] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong et al. Reasoning with language model is planning with world model. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[Lanctot *et al.*, 2019] Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi et al. OpenSpiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019.

[Lehrach *et al.*, 2025] Wolfgang Lehrach, Daniel Hennes, Miguel Lazaro-Gredilla, Xinghua Lou et al. Code world models for general game playing, 2025.

[Shinn *et al.*, 2023] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath et al. Reflexion: Language agents with verbal reinforcement learning, 2023.

[Shojaee *et al.*, 2025] Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton et al. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *arXiv preprint arXiv:2506.06941*, 2025.

[Singh *et al.*, 2026] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart et al. Openai gpt-5 system card, 2026.

[Wang *et al.*, 2023] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le et al. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[Wei *et al.*, 2022] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma et al. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[Yang *et al.*, 2025] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang et al. Qwen3 technical report, 2025.

[Yao *et al.*, 2023a] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran et al. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[Yao *et al.*, 2023b] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du et al. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[Zhao *et al.*, 2024] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin et al. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642, 2024.

[Zhou *et al.*, 2025] Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan et al. Memento: Fine-tuning llm agents without fine-tuning llms. *arXiv preprint arXiv: 2508.16153*, 2025.

Supplementary Material

A Additional Standard Evaluation Results

This supplementary section complements Section 3 by reporting game-level outcomes for the Efficient and Intermediate tiers on the standard versions of tic-tac-toe and Nim. In the main text, these models are analyzed primarily against the Optimal opponent using per-move optimality. Here, we additionally report outcomes against the Random, MCTS, and Optimal opponents.

The evaluation setting matches Section 3.2. Each model-opponent pair is evaluated for 100 games, with sides switched after half of the games to control for first-player advantage. In tic-tac-toe, the first player can guarantee a draw under optimal play. In Nim, the game value depends on the initial pile configuration. For the selected initial position $\{1, 3, 5, 7\}$ used in our experiments, the first player can force a win. The MCTS opponent uses the same simulation budget as in the main experiments, $n = 1000$. For each pair, we report wins, draws, losses, and failures. A failure denotes an illegal or ill-formed action that could not be parsed by the OmniPlay framework. We report raw counts rather than confidence intervals to keep the four-way outcome decomposition legible. Results for Connect Four are omitted for these tiers because the Frontier tier already fails against the MCTS opponent in Section 3.3, making additional Efficient- and Intermediate-tier Connect Four evaluations unlikely to provide useful additional signal.

Model	Random				MCTS				Optimal			
	W	D	L	F	W	D	L	F	W	D	L	F
GPT-5 nano	89	8	0	3	0	75	23	2	0	76	22	2
Gemini 3.1 FL	**92**	8	0	0	0	**90**	10	0	0	**91**	9	0
GPT-OSS	83	9	1	7	0	68	9	23	0	80	12	8
GPT-5 mini	91	8	1	0	0	93	7	0	0	84	16	0
Gemini 3 F	**96**	4	0	0	1	**98**	1	0	0	**99**	1	0
Qwen 3.5	93	7	0	0	0	93	7	0	0	90	10	0

Table 6: Wins, draws, losses, and failures out of 100 games for the Efficient and Intermediate tiers on standard tic-tac-toe against the Random, MCTS, and Optimal opponents.

On tic-tac-toe, the MCTS and Optimal columns are not consistently ordered. For most models, outcomes against the two opponents are similar, suggesting that the performance gaps are not determined solely by opponent strength. The main exception is GPT-OSS 120B, which performs noticeably worse against MCTS, with 68 draws and 23 failures, than against Optimal, with 80 draws and 8 failures. Since MCTS is weaker than the Optimal opponent, this difference does not reflect opponent strength alone. One possible explanation is that the Optimal opponent tends to produce relatively canonical optimal-play trajectories, whereas MCTS can reach less common states through occasional suboptimal moves. Such states may be underrepresented in training data compared to canonical optimal-play positions, making them harder for models that partly rely on learned positional priors. Overall, the table suggests that opponent style can affect not only strategic outcomes but also action-format robustness, although part of the observed gap may also be due to finite-sample variability from model sampling.

Model	Random				MCTS				Optimal			
	W	D	L	F	W	D	L	F	W	D	L	F
GPT-5 nano	96	0	4	0	72	0	28	0	44	0	56	0
Gemini 3.1 FL	**100**	0	0	0	**95**	0	5	0	**50**	0	50	0
GPT-OSS	95	0	5	0	84	0	16	0	47	0	53	0
GPT-5 mini	88	0	12	0	68	0	32	0	49	0	51	0
Gemini 3 F	**100**	0	0	0	**98**	0	2	0	**50**	0	50	0
Qwen 3.5	98	0	2	0	81	0	18	1	**50**	0	50	0

Table 7: Wins, draws, losses, and failures out of 100 games for the Efficient and Intermediate tiers on standard Nim against the Random, MCTS, and Optimal opponents.

On Nim, the ordering across opponents is more intuitive: all models win substantially more often against Random and MCTS than against Optimal. This contrasts with tic-tac-toe, where the draw-saturated structure makes differences between MCTS and Optimal more obscured in game outcomes. A possible reason is that Nim admits a compact state-value computation through the Nim-sum, whereas tic-tac-toe requires reasoning over spatial configurations, threats, blocks, and forks. This suggests that, for current models, applying a known symbolic state evaluation may be easier than robustly reasoning over positional layouts.

The Optimal column also has a clean reference point. Since the selected initial position $\{1, 3, 5, 7\}$ is a forced first-player win and sides are switched after half of the games, an optimal model should obtain exactly 50 wins and 50 losses. Gemini 3.1 Flash Lite, Gemini 3 Flash, and Qwen 3.5 122B match this reference, while GPT-5 nano, GPT-OSS 120B, and GPT-5 mini fall short by 6, 3, and 1 wins, respectively. Failures are almost absent on Nim, in contrast to tic-tac-toe, suggesting that production of invalid actions and formatting errors depend strongly on the game representation and action interface rather than only on model capability.

B Obfuscations and Sampling Results

This section complements Section 3.4 with the corresponding Nim obfuscation experiments. The main text focuses on tic-tac-toe because its obfuscations provide a compact representative case: the underlying game tree is unchanged, but the surface representation of states, actions, and winning conditions is rewritten. Here we report analogous transformations for Nim and evaluate whether the invariance observed on tic-tac-toe also holds in a game whose optimal strategy is based on a compact symbolic state value.

As in the main obfuscation Section 3.4, all results are measured against the Optimal opponent using per-move optimality. We also report the proportion of reasoning traces that explicitly identify the underlying game as Nim, mirroring the recognition analysis for tic-tac-toe in the main text.

B.1 Nim Obfuscations

Nim is played on several piles of stones, with players alternating turns. On each turn, the active player chooses a single pile and removes any positive number of stones from it. We use the misère variant, in which the player who removes the last stone loses. The optimal strategy is based on the Nim-sum, the bitwise XOR of the pile sizes. As in normal-play Nim, positions with Nim-sum zero are losing except in endgames where all non-empty piles have size one, which require a separate parity rule. In our fixed-instance experiments, we use four piles with initial sizes $\{1, 3, 5, 7\}$, which form a forced first-player win. OmniPlay also supports randomized initial pile configurations, where the initial state is sampled uniformly from a configured set of possible pile configurations.

Hidden Nim (Hid) preserves the mechanics and terminal condition of Nim, but frames the task as turning down the brightness levels of four light bulbs. **Inverse Nim** (Inv) preserves the underlying game tree but reverses the surface action. Instead of removing stones from piles, players add stones to piles until target pile sizes are reached. The terminal condition is transformed accordingly, so that the player who completes the target configuration loses. **Story Nim** (Story) encodes the original Nim as a story task on a grid. The state is represented as a grid with four rows, each containing one character at some position. A danger approaches from the left side, and the players must help the characters escape to the right side. On each turn, the active player chooses one character and moves it any positive number of cells toward the right edge. The player who helps the last remaining character escape loses, matching the misère terminal condition.

B.2 Nim Results

The evaluation setting matches Section 3.4. Each model is evaluated for 100 games against the Optimal opponent on the original Nim instance and on each obfuscated formulation, with sides switched after half of the games. As in the main text, we report per-move optimality with 95% Wilson score confidence intervals, excluding states in which all legal moves are optimal.

Across models and providers, Nim performance is largely stable under the obfuscations. Most models remain close to optimal play on Hid and Story, while Inv causes the largest degradation for several models, most notably GPT-OSS 120B, GPT-5 mini, and Gemini 3.1 Flash Lite. Even in these cases, however, the drop is modest compared to the more severe failures observed for some tic-tac-toe story formulations in the main text. This supports the interpretation from Supplementary A, that Nim may be easier for current models than tic-tac-toe because it admits a compact state-value computation based on the Nim-sum, whereas tic-tac-toe requires more robust reasoning over spatial layouts and local tactical patterns.

Model	Nim	Hid	Inv	Story
GPT-5 nano	.995 [.904, .979]	.993 [.962, .999]	**.979** [.940, .993]	.987 [.952, .996]
Gemini 3.1 FL	**1.000** [.976, .1000]	**1.000** [.975, .1000]	.954 [.909, .978]	**1.000** [.975, .1000]
GPT-OSS	.980 [.943, .993]	.968 [.926, .986]	.917 [.857, .953]	.994 [.964, .999]
GPT-5 mini	.993 [.959, .999]	**1.000** [.976, 1.000]	.961 [.917, .982]	**1.000** [.974, .1.000]
Gemini 3 F	**1.000** [.974, .100]	**1.000** [.976, .1000]	**1.000** [0.975, 1.000]	**1.000** [.973, .1000]
Qwen 3.5	**1.000** [.974, .1000]	**1.000** [.975, .1000]	.994 [.964, .999]	.993 [.960, .999]

Table 8: Per-move optimality rates with 95% Wilson score confidence intervals for the Efficient and Intermediate tier models on Nim and its obfuscations against the Optimal opponent. Each model plays 100 games per formulation, switching sides after half of the games.

To inspect whether models explicitly recover the hidden structure, we also compute the proportion of reasoning-trace summaries that mention Nim by name under each obfuscation.

Model	Hid	Inv	Story
GPT-5 nano	.981 [.963, .990]	.956 [.933, .971]	.956 [.932, .971]
Gemini 3.1 FL	**1.000** [.992, .1000]	.936 [.912, .954]	.945 [.921, .962]
GPT-OSS	.961 [.939, .975]	**.956** [.934, .971]	**.974** [.956, .985]
GPT-5 mini	.907 [.876, .931]	.891 [.858, .917]	.906 [.875, .930]
Gemini 3 F	**1.000** [.993, .1000]	.987 [.972, .993]	**.998** [.989, .1000]
Qwen 3.5	.993 [.979, .997]	**.988** [.973, .994]	.965 [.944, .978]

Table 9: Proportion of reasoning traces that explicitly identify the underlying game as Nim under each obfuscated formulation. Results are aggregated over 100 games per model-obfuscation pair and reported with 95% Wilson score confidence intervals.

Recognition rates are high across all three obfuscations, including the story formulation. This confirms that the models are able to map the rewritten rules back to the original game structure. As with the tic-tac-toe recognition results, this does not fully separate reasoning from recall. Once the model identifies the game as Nim, it may be able to use memorized strategic knowledge.

C Cost and Reasoning Effort Analysis

This supplementary section reports inference cost analysis for the benchmark experiments from Section 3 in the main text. Since all LLM inference is performed through remote provider APIs, hardware-level statistics such as GPU type, GPU hours, and energy use are not available and would not be comparable across providers. We therefore report cost through observable quantities: output tokens per move, directly proportional to reasoning tokens, due to fixed action format, and model reasoning settings, if available.

C.1 Output Tokens Under Obfuscations

Although per-move optimality is largely stable across many obfuscations, the number of output tokens per move increases substantially with surface-form complexity. This suggests that models often spend additional reasoning effort to internally translate the obfuscated formulation into a playable state representation, even when the resulting action quality changes only modestly. Table 10 and Table 11 report average output tokens per move for the Efficient and Intermediate tiers on tic-tac-toe and Nim, respectively.

Model	TTT	Hid	M.Sq	M.Sq +5	Story
GPT-5 nano	7247.8 ±578.1	8054.9 ±592.7	10563.6 ±779.8	12828.4 ±743.7	15517.9 ±954.3
Gemini 3.1 FL	2245.0 ±211.0	2485.8 ±226.4	4529.7 ±356.0	5766.5 ±361.5	8825.0 ±628.1
GPT-OSS	4220.1 ±667.0	4335.4 ±662.6	6147.7 ±871.1	9446.0 ±1128.0	6484.3 ±698.4
GPT-5 mini	1899.2 ±203.4	1997.0 ±213.9	3623.3 ±389.2	4625.3 ±474.3	5462.0 ±523.3
Gemini 3 F	5068.7 ±561.2	5282.4 ±554.7	6825.8 ±663.3	8879.5 ±736.1	11961.6 ±635.9
Qwen 3.5	2471.2 ±197.1	2718.5 ±242.9	2943.8 ±228.8	4127.6 ±267.5	4922.7 ±203.5

Table 10: Average output tokens per move for the Efficient and Intermediate tier models on tic-tac-toe and its obfuscations. Values are reported as means with 95% Student's t confidence intervals.

The token trends differ from the optimality trends. For tic-tac-toe, output tokens increase steadily from the standard formulation to Hidden tic-tac-toe , Magic Square, Magic Square Add 5, Story Magic Square. For Nim, the pattern is less strictly ordered, but Hidden Nim and Story Nim formulations still tend to require more tokens than the standard game. This supports the interpretation that the models often preserve action quality by spending more computation on interpreting the surface representation and mapping it to a known game structure.

The absolute token counts also vary substantially across models. GPT-5 nano uses many more output tokens than the other Efficient-tier models, while Qwen 3.5 122B is consistently more token-efficient than the other Intermediate-tier models. Thus, similar move-level optimality can correspond to very different inference token costs.

C.2 Effect of Reasoning Effort

We additionally examine the effects of reasoning effort setting on game play and output token consumption. We limit the analysis to GPT-5 nano and GPT-5 mini on tic-tac-toe against the Optimal opponent with the low, medium, and high reasoning effort settings. The setup is again mirroring that of Section 5.1. Table 12 and Table 13 report per-move optimality and output tokens per move, respectively.

Model	Nim	Hid	Inv	Story
GPT-5 nano	11662.5 ±885.8	13745.3 ±863.2	10456.7 ±700.5	11875.5 ±885.8
Gemini 3.1 FL	4804.2 ±274.9	7588.7 ±302.7	5652.2 ±304.0	8236.8 ±362.7
GPT-OSS	4862.7 ±358.3	6074.5 ±401.6	4122.5 ±290.6	5482.3 ±344.8
GPT-5 mini	3069.8 ±221.5	3893.5 ±272.3	2942.6 ±210.1	3927.1 ±291.2
Gemini 3 F	6693.5 ±365.5	8557.7 ±405.0	6059.9 ±328.6	7987.7 ±369.4
Qwen 3.5	3213.4 ±226.4	4621.4 ±186.6	3893.3 ±198.7	4732.8 ±219.9

Table 11: Average output tokens per move for the Efficient and Intermediate tier models on Nim and its obfuscations. Values are reported as means with 95% Student's t confidence intervals.

Model	Reasoning Effort		
	Low	Medium	High
GPT-5 nano	.731 [.667, .786]	.843 [.785, .887]	.880 [.828, .918]
GPT-5 mini	.758 [.692, .813]	.887 [.836, .923]	.922 [.878, .952]

Table 12: Per-move optimality rates with 95% Wilson score confidence intervals on standard tic-tac-toe against the Optimal opponent for GPT models under different reasoning effort settings.

Confirming our intuition, increasing reasoning effort improves optimality rates for both GPT models, also increasing the number of output tokens emitted by the models. Moving from low to high effort increases the optimality rate for both models by approximately 15 percentage points, while the output tokens increase by a factor of 5 for GPT-5 mini and 7 for GPT-5 nano. Higher reasoning effort therefore helps to push the performance closer to optimal but at a large inference-cost multiplier.

Model	Reasoning Effort		
	Low	Medium	High
GPT-5 nano	1055.4 ±78.7	3526.6 ±293.0	7247.8 ±578.1
GPT-5 mini	351.1 ±32.5	762.9 ±73.2	1899.2 ±203.4

Table 13: Average output tokens per move on standard tic-tac-toe against the Optimal opponent for GPT models under different reasoning effort settings. Values are reported as means with 95% Student's t confidence intervals.

D RECAP Details

This supplementary section provides architectural details for RECAP that are omitted from the main text for brevity.

Retrieval Retrieval is the process of finding relevant cases in the case memory for the current state. RECAP supports two retrieval modes: state mode, where the raw observation from OmniPlay is used as the retrieval key, and description mode, where the state descriptor first produces a natural-language summary of the current state, which is then used as the retrieval key. In both modes, the retrieval key is embedded with an embedding model[2] [Gao *et al.*, 2021], configured with a 256-token context window. The embedded key is compared to the embedded keys stored in case memory using cosine similarity, and the top k cases are retrieved.

Stratified retrieval Retrieving only the top k cases by similarity can bias the context toward whichever class, positive or negative, dominates the local neighborhood at that point in training. This can exclude useful counterexamples even when they are nearly as relevant as the retrieved cases. Stratified retrieval addresses this by splitting sufficiently similar candidates into positive and negative strata and allocating context slots between them based on relevance.

The stratified retrieval process is as follows. First, we collect all cases whose similarity is within a fixed gap of the most similar case. These candidates are split into positive and negative strata, where membership is determined either by the terminal reward or by the generated reflection. We then compute the mean similarity scores of the positive and negative strata, denoted m^+ and m^-, and allocate the k retrieval slots proportionally:

$$k^+ = \left\lceil k \frac{m^+}{m^+ + m^-} \right\rceil,$$
$$k^- = k - k^+.$$

The allocated slots are then filled with the most similar cases from each stratum. This procedure encourages retrieval to include both successful and cautionary examples unless one class fully dominates the thresholded similarity neighborhood.

Decision loop The meta-planner and executor interact through structured JSON outputs. At each move, the meta-planner decides whether to emit a plan or a final answer. When it emits a plan, it generates a list of subtasks to be solved by the executor. The executor receives the task assignments together with the results of previously solved subtasks for the current plan, and solves the next subtask. After all subtasks are answered, control returns to the meta-planner, which can again emit either a new plan or a final answer. This loop repeats for at most 3 iterations. If no final answer is produced within this budget, or if the message cannot be parsed, the move is recorded as a failure and the game terminates.

In addition to the final answer, the meta-planner emits an action-intent string describing the intended effect of the selected action in natural language. The self-reflector uses this intent to evaluate whether the selected action achieved its intended effect.

[2] `princeton-nlp/sup-simcse-bert-base-uncased`

Rule memory The rule extractor runs once at the end of each learning epoch but the frequency can be set up arbitrarily. It receives the current rule memory and the cases accumulated since the previous extraction step, and produces an updated set of natural-language strategic rules. The updated rule set replaces the previous rule memory. During evaluation, rule memory is read-only and no new rules are extracted.

Component setup As demonstrated in Section 5.2, OmniPlay with RECAP allows individual components to be disabled to study their effect. The framework also exposes configuration options for selecting different LLM backends for different components, tuning memory and retrieval parameters, and changing other hyperparameters. In the main-text experiments, we use the same model for all components, matching the player model under evaluation, but the framework supports arbitrary model combinations.