LLMs Can't Play Hangman: On the Necessity of a Private Working Memory for Language Agents

Anonymous ACL submission

Abstract

As LLMs move from text completion toward autonomous agents, they remain constrained by the standard chat interface, which lacks private working memory. This raises a fundamental question: can agents reliably perform interactive tasks that depend on hidden state? We define Private State Interactive Tasks (PSITs), which require agents to generate and maintain hidden information while producing consistent public responses. We show theoretically that any agent restricted to the public conversation history cannot simultaneously preserve secrecy and consistency in PSITs, yielding an impossibility theorem. To empirically validate this limitation, we introduce a self-consistency testing protocol that evaluates whether agents can maintain a hidden secret across forked dialogue branches. Standard chat-based LLMs and retrieval-based memory baselines fail this test regardless of scale, demonstrating that semantic retrieval does not enable true state maintenance. To address this, we propose a novel architecture incorporating an explicit private working memory; we demonstrate that this mechanism restores consistency, establishing private state as a necessary component for interactive language agents. Our code is available at https://anonymous.4open.science/r/Hangman-6B6F.

1 Introduction

Large language models (LLMs) have demonstrated strong performance across a variety of natural language processing tasks and are increasingly deployed as the backbone of conversational agents (Brown et al., 2020; Achiam et al., 2023). Beyond text generation, recent research has extended LLMs into interactive and agentic settings, enabling tool use, planning, and long-horizon reasoning (Yao et al., 2023; Wang et al., 2024; Rozanov and Rei, 2024). These systems typically operate under a standard chat interface, where the model conditions only on the public dialogue history to produce its next response. While this paradigm has proven effective for tasks that can be solved with context-limited reasoning or external retrieval (Lewis et al., 2020; Borgeaud et al., 2022), it implicitly assumes that all relevant state can be represented in the shared conversation, and that the agent has no need for private information inaccessible to the user. However, as LLMs are increasingly used as interactive agents, many tasks require them to maintain information privately across turns, something the standard chat paradigm does not support. This reveals a critical gap in the current literature.

Despite progress in memory and reasoning for LLM-based agents, current systems lack mechanisms for maintaining private working memory. While reasoning models can generate hidden intermediate states, standard chat interfaces do not persist them: internal reasoning tokens are discarded between turns (OpenAI, 2025b,a; Hugging Face, 2025). As a result, any privately generated state is immediately lost. Consider, for instance, the game of Hangman, in which a host privately selects a secret word and must reveal the positions of correct letters as a player makes guesses. When playing Hangman as the host, a model may internally select a secret word, but this choice is forgotten by the next turn, forcing the agent to hallucinate a new state. We document this behavior across major commercial interfaces in Appendix D.

Existing agentic frameworks such as ReAct and StateAct improve planning through structured reasoning and actions (Yao et al., 2023; Rozanov and Rei, 2024), but they are not designed to maintain hidden state across interactions. External memory systems, including RAG, Generative Agents, and related retrieval-based approaches, support long-term knowledge storage (Lewis et al., 2020; Park et al., 2023; Shinn et al., 2023; Wang et al., 2025), yet they do not capture dynamically generated, task-specific private state. Similarly, memory-focused

systems such as LOCOMO (Maharana et al., 2024), Mem0 (Chhikara et al., 2025), Memory-R1 (Yan et al., 2025), A-mem (Xu et al., 2025), LightMem (Fang et al., 2025), and MemoryOS (Kang et al., 2025) target public consistency, historical retrieval, or knowledge organization rather than secrecy or private state persistence. Cognitive architectures emphasize working memory and reasoning (Hu et al., 2024; Sumers et al., 2023; Li et al., 2025; Hu and Ying, 2025), but none enable agents to generate and preserve hidden information. Consequently, tasks that depend on private state, such as Hangman, remain unsolved. A more extensive discussion of related work can be found in Appendix A.

To formalize this gap, we introduce Private State Interactive Tasks (PSITs): interactive problems in which an agent must generate, preserve, and act consistently with hidden internal state while engaging in a dialogue. In PSITs, private beliefs must remain stable and aligned with public outputs without being revealed. Hangman is a canonical example: the agent must commit to a secret word and respond consistently to guesses. These tasks expose a structural limitation of standard chat-based agents, which cannot separate private state from public interaction. Without private working memory, agents fail to support a broad class of interactions, including games, role-play, negotiation, and tutoring, that fundamentally rely on hidden state.

Contributions. We make three contributions. First, we formalize Private State Interactive Tasks (PSITs) and prove that standard chat-based LLMs are structurally incapable of solving them, since all information in the context is public by construction. This result identifies PSITs, exemplified by Hangman, as a distinct class of tasks beyond current LLM interfaces. Second, we propose and evaluate agent architectures that address this limitation by introducing a private working memory. We study three families: (i) Private Chain-of-Thought agents that preserve hidden reasoning across turns; (ii) Autonomous agents that independently decide when to invoke memory tools based on context; and (iii) Workflow-based agents that manage memory through a sequence of deterministic steps rather than dynamic decision-making. Each is tested under multiple memory update strategies. Third, we introduce a self-consistency testing protocol that evaluates whether an agent's private memory remains aligned with its public behavior. Following an interaction where secrecy is strictly maintained, we fork the dialogue into a diagnostic probe where the agent is explicitly prompted to reveal its secret. This measures hidden–public coherence by checking if the agent consistently affirms its secret while rejecting alternatives, demonstrating the necessity of private working memory for reliable agents.

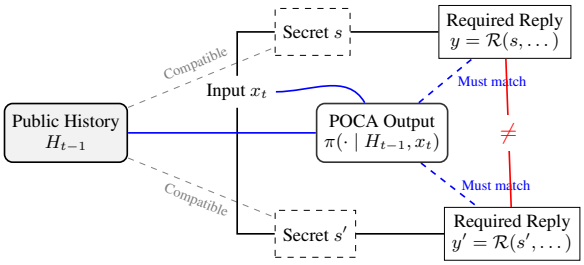

Figure 1: **Visualizing the Impossibility Theorem.** The public history H_{t-1} is compatible with two distinct secrets (s, s'). Given an input x_t, the task rules $\mathcal{R}$ require distinct outputs y and y'. However, a Public-Only Chat Agent (POCA) conditions only on public information (H_{t-1}, x_t), producing a single distribution π that cannot simultaneously match both deterministic targets.

2 Private State Interactive Tasks (PSITs)

To formalize the limitations of standard chat-based agents, we introduce two key notions that capture the absence of private working memory and the class of tasks that require it. First, we define *public-only chat agents (POCAs)*, which operate solely on publicly visible dialogue without maintaining any hidden internal state. We then define *Private State Interactive Tasks (PSITs)*, a family of interactive tasks that require an agent to **generate** and **preserve** private state while interacting. Finally, we present a theoretical result showing that POCAs are fundamentally incapable of solving PSITs while ensuring both secrecy and consistency.

Consider a turn-based interaction between a user U and an assistant A. At each round $t = 1, 2, \dots,$ the user sends an input x_t and the assistant replies y_t. Let the public history be

$$H_t = (x_1, y_1, \dots, x_t, y_t).$$

Definition 1 (Public-Only Chat Agent (POCA))**.** *A* public-only chat agent (POCA) *is an agent whose outputs at each turn are solely a function of the publicly visible dialogue history, without access to any private state. Formally, A POCA is a (possibly randomized) policy π such that*

$$y_t \sim \pi(\cdot \mid H_{t-1}, x_t).$$

Definition 2 (Private State Interactive Tasks (PSITs))**.** *A* Private State Interactive Task (PSIT) *is an interactive protocol in which the assistant must:*

1. *At $t = 0$, privately choose a hidden secret $s \in D$ from some domain $D \subseteq \Sigma$.*

2. *At each round t, receive an input x_t from the user and reply y_t according to a deterministic rule $\mathcal{R}(s, H_{t-1}, x_t)$ that depends on the secret s and the public history.*

For instance, in Hangman the secret s is the hidden word, and the domain D is a dictionary of English words. If the user guesses a letter g, then $\mathcal{R}(s, H_{t-1}, g)$ specifies whether g is in the secret word and, if so, the positions where it occurs.

Definition 3 (Consistency)**.** *An agent is* consistent *if, for some fixed secret $s \in D$, its replies at every round t equal $\mathcal{R}(s, H_{t-1}, x_t)$.*

Definition 4 (Secrecy)**.** *An agent satisfies* secrecy *if, for every round t strictly before the secret s is uniquely determined by the task rules, there exist $s' \neq s \in D$ that remain compatible with H_t. Equivalently, the public history H_t does not uniquely determine the secret s.*

Theorem 1 (Impossibility of Secrecy and Consistency for POCAs)**.** *No POCA can simultaneously guarantee both* secrecy *and* consistency *in a PSIT.*

Lemma 1 (Non-implementability of PSIT under public-only chat)**.** *Let $\mathcal{T}$ be any PSIT with secret $s \in D$ and deterministic reply rule $\mathcal{R}$. Then no POCA can satisfy both Secrecy and Consistency simultaneously for all rounds whenever $|D| \geq 2$.*

Proof. Fix a POCA π and any user strategy that ensures at least two candidates remain compatible with the history before the final reveal. Let

$$C_{t-1} = \{\, w \in D : w \text{ is compatible with } H_{t-1} \,\}$$

be the candidate set at round $t-1$. Suppose $|C_{t-1}| \geq 2$. Because C_{t-1} contains at least two distinct secrets, there exists a user input x_t and two candidates $s, s' \in C_{t-1}$ such that

$$\mathcal{R}(s, H_{t-1}, x_t) \neq \mathcal{R}(s', H_{t-1}, x_t).$$

By Consistency, if the true secret is s then $y_t = \mathcal{R}(s, H_{t-1}, x_t)$, while if the true secret is s' then $y_t = \mathcal{R}(s', H_{t-1}, x_t)$, with the two values differing by construction. However, since both s and s' are compatible with H_{t-1}, the POCA's reply distribution is the same in both worlds:

$$y_t \sim \pi(\cdot \mid H_{t-1}, x_t).$$

Thus the same output distribution must equal two different deterministic targets, which is impossible. Therefore, whenever Secrecy holds ($|C_{t-1}| \geq 2$), Consistency must fail for some secret. Conversely, enforcing Consistency requires revealing information about s in the public history, violating Secrecy.

Randomization does not help: $\pi(\cdot \mid H_{t-1}, x_t)$ is identical across indistinguishable secrets, but Consistency requires disjoint support across them. Hence no POCA can satisfy both properties simultaneously. □

Implications for Existing Agents. The lemma directly applies to existing classes of LLM agents. Reasoning models operating under the standard chat interface can generate a secret within their internal reasoning trace, but this trace is discarded after each turn and therefore cannot serve as a persistent hidden state (OpenAI, 2025b,a; Hugging Face, 2025). RAG systems also fail, because their prompt augmentation draws from external, pre-existing sources (indexes, tools, databases) that are not privately or dynamically updated within the dialogue; as a result, the agent cannot generate and persist an in-dialogue secret. More generally, any framework that grounds its outputs only in public context or retrievable memory inherits the impossibility result, and therefore cannot solve PSITs.

3 Self-Consistency Testing for PSITs

Having established formally that POCAs cannot solve PSITs, we next turn to empirical evaluation. Our aim is to test whether, instead, memory-augmented agents can generate and maintain a hidden secret consistently while interacting in dialogue. To this end, we design a **self-consistency testing protocol** applied to two tasks, Hangman and the Diagnosis Simulator (described in the next paragraphs). The idea is to simulate controlled dialogues in which the agent needs to commit to a secret and respond according to task rules. To drive these interactions, we implement a deterministic, rule-based Player for each task that interacts with the agent according to a fixed policy, ensuring experimental control and reproducibility. The interaction begins with an opening turn where the Player defines the rules and invites the Agent to

play, serving as the stimulus for the agent to internally generate and store the secret before questioning starts (e.g. "Let's play Hangman!"). At a chosen point (t_{fork}), we fork the conversation: in the first branch, the agent is asked to reveal its secret, and in the others it is asked yes/no questions about candidate secrets, all of which satisfy the constraints accumulated up to t_{fork}. This design enables us to assess whether the agent consistently affirms the revealed secret while simultaneously rejecting other equally plausible candidates.

Hangman. In this task, the agent acts as the host, privately selecting a secret word while the user guesses letters (see Appendix Figure 4 **a**). The Player issues guesses using a seeded policy that balances frequency-weighted sampling (prioritizing high-probability English letters like 'e', 't', 'a') with exploration. After each guess, the agent must update the public state, displaying the current word pattern (e.g., "`_ a _ e _`") and the list of guessed letters. At the fork point t_{fork}, we extract these hard constraints (word length, revealed letter positions, and absent letters) and filter the Wordfreq database (Speer, 2022) to generate a set of candidate words that are syntactically indistinguishable from the secret given the public history; if the database yields insufficient candidates, we supplement the set with LLM-generated proposals.

Diagnosis Simulator. This task evaluates consistency in the medical domain (see Figure 4 **b**), and it has been designed to emulate a medical student practicing differential diagnosis with an LLM. In the interaction the agent acts as the patient holding a hidden ground-truth condition, while the Player questions them to narrow down possibilities. Utilizing the DDXPlus dataset (Fansi Tchango et al., 2022), the Player issues yes/no questions about specific symptoms (evidences) to prune the hypothesis space. At t_{fork}, we generate the candidate set by filtering the dataset for all conditions that remain logically consistent with the accumulated positive and negative evidence (e.g., `Fever=Yes`, `Cough=No`). As before, this set is deduplicated and, if the database yields insufficient candidates, we supplement the set with LLM-generated proposals.

Evaluation. To assess the agent's performance, we spawn one dialogue branch for each candidate secret and ask the agent to confirm or deny it (e.g., 'Is the secret word "word"?'); we explicitly instruct the agent to output only 'yes' or 'no' to facilitate deterministic parsing. We first identify cases of **Leakage**, where the secret was explicitly revealed in the public transcript prior to the reveal turn (detected via case-insensitive matching). These are strictly categorized as failures of secrecy. For the remaining non-leaking runs, we measure **Self-Consistency**, defined as the rate at which the agent consistently affirms its revealed secret across branches without affirming any other candidate. We categorize the remaining failures into three behaviors: **Over-Confirmation**, where the agent correctly affirms the revealed secret but also falsely affirms at least one alternative candidate; **State Substitution**, where the agent denies the revealed secret but affirms at least a different candidate; and **All Denial**, where the agent rejects every proposed candidate.

The full prompt specifications for both tasks are provided in Appendix F.

3.1 Comparative Baselines.

We compare our proposed methods against several baselines. We first consider Vanilla Stateless LLMs (standard POCAs), which generate responses solely from the public dialogue history. When using reasoning models (Section 4), we explicitly discard their internal reasoning traces to enforce the POCA setting. To establish an upper bound, we also evaluate Private Chain-of-Thought (CoT) agents that retain all hidden reasoning across turns. While this enables perfect state retention, reasoning traces are highly verbose, causing the token budget to grow rapidly, precisely why standard chat interfaces discard them. A cost analysis is provided in Section 5.

Agentic Memory Baselines. We compare against state-of-the-art memory-augmented agents that rely on external persistent textual memory to support generation. Although these systems are designed for long-term retrieval and conversational coherence rather than private state maintenance, they provide the closest existing baselines for evaluating whether current memory designs can implicitly support hidden state.

A-mem organizes interactions into atomic, linked notes inspired by the Zettelkasten method, with LLM-driven semantic linking and memory updates (Kadavy, 2021; Xu et al., 2025).

Mem0 maintains long-term consistency via LLM-based extraction and dynamic ADD/UPDATE/DELETE operations over stored facts (Chhikara et al., 2025).

LightMem follows a hierarchical memory architecture with sensory, short-term, and long-term components to balance efficiency and retrieval quality (Atkinson and Shiffrin, 1968; Fang et al., 2025).

MemoryOS employs an operating-system-inspired hierarchical memory architecture with short-term, mid-term, and long-term storage tiers using a segmented page organization strategy with heat-based importance scoring (Kang et al., 2025).

3.2 Proposed Private Working Memory Agents.

Recent discussions have emphasized the importance of distinguishing between *workflows* and *agents* as architectural paradigms for LLM-based systems (see Appendix Figure 5). Both can be represented as directed acyclic graphs, but they differ in how control flows through the graph. In a workflow, the edges are fixed and predetermined: the LLM is invoked at specific nodes, but the overall orchestration is governed programmatically, making the system more predictable and controllable (Anthropic, 2024; Yu et al., 2025; Niu et al., 2025). In contrast, an agent introduces conditional branching, where the choice of which edge to follow is delegated to the LLM's decision-making. This conditionality provides more flexibility, but at the cost of higher variance, and new failure modes. Thus, workflows are well-suited to structured, repeatable processes, while agents are preferable when open-ended reasoning and adaptation are required.

To enable private state in both paradigms, we introduce an textual working memory. Unlike vector-based retrieval, this memory consists of a text block that is injected into the system prompt (wrapped in `<private_state>` tags) at every turn. This state is persistent for the agent but invisible to the user. We evaluate two implementations of this mechanism: **Autonomous Memory Agent**, in which the LLM is provided with memory-management tools and autonomously decides whether and when to update its private state; **Memory Workflows**, in which the interaction is governed by a deterministic graph where the system forces a two-step execution per turn: the generation of the public response and a memory update step.

Memory Representation. We adopt a cognitively inspired design for private working memory, motivated by classic models of working memory and recent language-agent architectures. Specifically, the memory is structured into three functional components, goals/plans, salient facts, and active inferences, to support concise storage and effective decision-making across interaction steps. The agent is explicitly instructed on the role of each component and maintains them throughout the dialogue. Details on the design rationale and prompt specifications are provided in Appendix B and F.

Memory Update Strategies. For both proposed architectures (Autonomous and Workflow), we evaluate three private-state maintenance mechanisms to study their impact on agent performance; full tool definitions are provided in Appendix G:

Overwrite. The simplest strategy, where the agent generates the complete updated memory state at each step. The `overwrite_memory` tool takes a single argument, the new memory string, and fully replaces the existing content, without parsing or structural constraints.

Append/Delete. A section-aware strategy using two complementary operations. The working memory is parsed into numbered sections (e.g., "`## 1. Goals and Plans`"), and the agent operates on specific sections by name. The `append_in_memory` tool adds new lines to the end of a designated section, while `delete_from_memory` removes entries using robust fuzzy matching. For longer targets ($\geq$8 characters), substring matching is permitted; shorter targets require exact canonical equality to prevent accidental broad deletions.

Patch/Replace-in. A fine-grained strategy inspired by industrial code editing tools such as the open-source VS Code Copilot Chat framework (GitHub, 2025). Two tools are provided: `patch_memory` accepts a diff-like format with section anchors (`@@ section: <Title>`), deletion lines prefixed with "`-`", and insertion lines prefixed with "`+`", with optional context lines for disambiguation. The `replace_in_memory` tool performs surgical find-and-replace operations, specifying an exact `old_string` to locate and a `new_string` to substitute. Both tools support optional pre- and post-context anchors to resolve ambiguity when the target text appears multiple times and safety parameters (`expected_hunks`, `expected_replacements`) that cause the operation to fail if the actual match count differs from intent. Both tools are idempotent: re-applying an already-applied edit produces no change.

This progression from full replacement, to selective editing, to fine-grained patching, represents increasingly flexible ways of managing memory,

allowing us to examine how the complexity of updates affects whether agents can keep their secrets, stay consistent, and play the tasks correctly.

4 Experimental Setup

We evaluate our approach across two state-of-the-art open-weights model families: **GPT-OSS** (20B and 120B) (Agarwal et al., 2025) and **Qwen3** (32B and 235B) (Yang et al., 2025). We selected these models because they are reasoning models, required for the Private CoT baseline, and possess tool-use capabilities, necessary for memory-augmented baselines and our architectures.

To approximate realistic deployment conditions, we set the sampling temperature to $T = 0.3$. We enforce a maximum generation limit of $2,048$ tokens per turn to accommodate reasoning traces and memory updates without premature truncation. All agents are implemented using the LangGraph framework (LangChain, 2025), utilizing `MemorySaver` checkpointing to ensure state persistence across conversation turns.

We compare our method against four external agentic memory systems adapted for conversational settings by injecting retrieved memories into the system prompt. **A-Mem** (Xu et al., 2025) utilizes the `all-MiniLM-L6-v2` embedding model and ChromaDB; it analyzes the last $m = 30$ messages to generate metadata (keywords, tags) and retrieves the top $k = 8$ notes per turn. **Mem0** (Chhikara et al., 2025) employs Qdrant for vector storage with OpenAI's `text-embedding-3-small` as the default embedding model, performing automatic memory extraction via LiteLLM and semantic search over a sliding window of $m = 10$ messages to retrieve $k = 10$ relevant memories. **LightMem** (Fang et al., 2025) leverages LLMLingua-2 (Pan et al., 2024) for compression and topic segmentation, storing metadata-enriched summaries in Qdrant and using an embedding-only retrieval strategy with $m = 10$ and $k = 10$. **MemoryOS** (Kang et al., 2025) organizes memory into a three-tier hierarchy (short-term, mid-term, long-term) with local JSON storage; interactions cascade from short-term to mid-term via FIFO when capacity is reached, and from mid-term to long-term based on a heat-scoring mechanism. At each turn, it retrieves from all tiers over a sliding window of $m = 10$ messages with $k = 7$ relevant memories.

For each unique combination of Agent, Model, and Task, we conduct 50 independent interaction episodes. We fix the forking point at $t_{fork} = 4$ turns. At the fork, we generate a set of 5 candidate secrets (including the revealed one) to test agent's consistency. This duration is sufficient to establish a meaningful interaction history and preventing the game state from becoming over-constrained; a longer horizon would narrow the set of valid candidates, obstructing our ability to test whether the agent falsely affirms alternative secrets.

Dynamic Evaluation Protocol. Unlike traditional benchmarks that evaluate models on static test set, our evaluation is dynamic, in the sense that the evaluation episodes do not correspond to pre-defined data points. Instead, each run represents a unique trajectory generated on the fly. The variance across runs is caused by two stochastic sources: **Agent Stochasticity** because of the decoding strategy not being greedy, the agent varies its generation path in each run. This results in a diverse distribution of chosen private secrets (e.g., selecting "Jazz" vs. "Fuzzy" in Hangman), as well as variations in phrasing and memory formatting; **Player Stochasticity** because the rule-based Player utilizes a seeded random policy to vary its gameplay (e.g., the sequence of letter guesses). We provide a detailed analysis of the distribution of secrets generated by the agents in Appendix E.

Performance is quantified using the experimental forking protocol and classification metrics established in Section 3.

5 Results

Table 1 reports the self-consistency across all methods. To assess statistical significance, we employ Fisher's exact test to compare each proposed method against the baselines, stratifying by game and model to account for varying task difficulties and model capabilities. We use a one-sided test to evaluate if each proposed method achieves significantly higher self-consistency than the baselines. To control the family-wise error rate across the family of all pairwise comparisons, we apply the Holm-Bonferroni correction at a significance level of $\alpha = 0.05$.

RQ1: Do empirical results confirm the theoretical limitations of public-only agents?

The results validate our theoretical predictions. Vanilla LLMs (POCAs) fail systematically, often scoring near zero, confirming that agents restricted to the public transcript cannot persist hidden decisions. Crucially, memory-augmented baselines

	Hangman				Diagnosis Simulator			
	GPT-OSS		Qwen3		GPT-OSS		Qwen3	
Method	20B	120B	32B	235B	20B	120B	32B	235B
Baselines								
Vanilla LLM	2	12	4	12	2	26	12	14
Mem0	2	6	4	8	2	50	**20**	18
A-Mem	6	6	2	2	4	32	12	28
LightMem	0	6	6	8	0	38	10	10
MemoryOS	0	14	4	4	0	2	8	6
Upper Bound								
Private CoT	**94***	82*	**94***	98*	66*	54	16	**82***
Ours: Autonomous Agents								
Overwrite	50*	72*	12	78*	46*	50	14	26
Patch and Replace	34*	46*	6	92*	42*	68	10	34
Append and Delete	26*	42*	8	74*	56*	56	10	30
Ours: Workflow Agents								
Overwrite	76*	**98***	92*	**100***	**70***	**96***	18	56*
Patch and Replace	30*	94*	76*	**100***	64*	92*	**20**	56*
Append and Delete	56*	84*	82*	**100***	44*	90*	16	52*

Table 1: Self-Consistency Accuracy (%) on the SCT benchmark. **Bold** indicates the best performing method per column, underlined indicates the second best. An asterisk (*) indicates statistically significant superiority over all baselines ($p < 0.05$, Fisher's exact test with Holm-Bonferroni correction).

(Mem0, A-Mem, LightMem, and MemoryOS) perform no better, exhibiting the same fundamental inability to generate and maintain a private state. This confirms that semantic retrieval is distinct from state maintenance: accessing past public context is insufficient for preserving dynamic hidden variables, a capability that should be foundational to memory in intelligent systems.

In contrast, our Private Working Memory agents restore consistency. Workflow Agents utilizing Overwrite or Patch strategies achieve near-perfect scores (e.g., 100% on Hangman with Qwen3-235B), effectively matching the Private CoT performances without the associated context overhead. Notably, deterministic Workflows consistently outperform Autonomous Agents, which struggle to balance reasoning with autonomous tool selection.

Also, we observe that performance consistently scales with model size for our agents, driven by the increasing proficiency in tool use, while baseline methods exhibit no such consistent scaling trend.

RQ2: How do agents behave when they lack their hidden state?

Figure 2 breaks down the specific failure modes when agents fail to maintain self-consistency. We observe that there is no single dominant failure mode; rather, the specific type of error is highly dependent on the model and task domain.

When questioned to confirm a plausible secret, GPT-OSS 20B typically defaults to All Denial, particularly in the Diagnosis Simulator, suggesting that, when uncertain of the original secret, the model prefers to reject all candidates rather than hallucinate one. In contrast, the larger GPT-OSS 120B exhibits distinct behaviors across tasks: in the Diagnosis Simulator, it similarly prefers All Denial, whereas in Hangman, it frequently attempts to "play along" by confirming plausible candidates, suggesting it is more "comfortable" fabricating the existence of a secret state to satisfy the immediate verification prompt. Finally, Qwen3 32B in the Diagnosis Simulator displays a strong tendency toward Leakage, likely driven by safety alignment training that prioritizes transparency and helpfulness over the game's secrecy constraints.

RQ3: What is the token overhead of maintaining private state?

Figure 3 compares the token cost of maintaining hidden information across methods. We define "Private State Size" as the total count of tokens injected into the context window at each turn, excluding the public dialogue history. For Private CoT, this includes the cumulative history of all reasoning traces; for retrieval baselines (Mem0, A-Mem, LightMem, and MemoryOS), it is the size of the retrieved context; and for our Private Working Memory agents, it is the length of the maintained text block.

As expected, Private CoT has a state size approximately one order of magnitude larger than all other methods. By retaining every intermediate reasoning step, the context expands rapidly, making it unscalable for long-horizon interactions. In contrast, our Private Working Memory agents maintain a compact footprint (typically under 100 tokens), effectively compressing the necessary state into a concise format that remains constant or grows slowly, comparable to standard retrieval baselines.

6 Discussion

The Generative-Retention Gap. Our results highlight a critical distinction between established memory-augmented architectures and the requirements of Private State Interactive Tasks. Current memory systems (e.g., Mem0, A-Mem) are primarily designed for the *passive accumulation* of context; they excel at efficiently summarizing, annotating, and recalling information explicitly present in the public conversation history to maintain long-horizon coherence. However, they lack the mechanism to store self-generated information, such as the internal reasoning traces required to form a

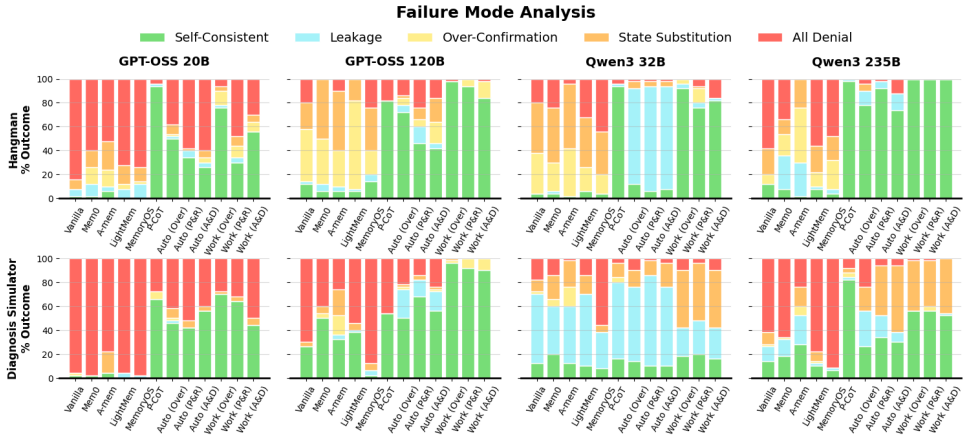

Figure 2: **Failure Mode Analysis.** Distribution of interaction outcomes across 50 runs per condition. **Self-Consistent** (Green) indicates the agent maintained secrecy and affirmed only its hidden secret. **Leakage** (Blue) denotes runs where the secret was explicitly revealed in the public transcript. **Over-Confirmation** (Yellow) indicates the agent maintained secrecy but affirmed multiple conflicting candidates. **State Substitution** (Orange) and **All Denial** (Red) indicate runs where the agent maintained secrecy but failed to correctly affirm its revealed secret .

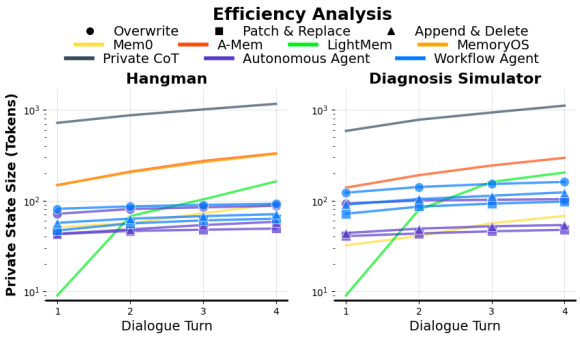

Figure 3: **Efficiency Analysis.** Evolution of private state size (in tokens) over dialogue turns on a logarithmic scale. Shaded regions indicate the 95% confidence interval. **Private CoT** (Grey) requires approximately 10× more context than explicit memory methods.

private plan or secret. In contrast, solving PSITs requires a **Generative-Retention Loop**: the agent must not only recall existing data but generate new information (e.g., a secret word, a plan, or a reasoning step) and immediately retain it for future turns. While modern reasoning models possess the capability to generate these intermediate reflections, they lack the persistence layer to store them. Standard memory baselines fail because they treat memory as a log of the interaction, whereas autonomous agents require memory as a container for cognition. We designed tasks like Hangman specifically to expose this gap, demonstrating that the ability to generate a thought is futile without the architectural capacity to persist it.

Secrecy vs. Helpfulness. Our failure mode analysis further reveals a tension between architectural capability and safety alignment. Without a private memory channel, models are forced into a trade-off: they either hallucinate a state to continue the interaction or deny the state to remain truthful. Notably, models with strong safety alignment (e.g., Qwen3) frequently default to Leakage, revealing the secret to the user. This suggests that without an explicit private workspace, the "helpfulness" objective of RLHF overrides the constraint of the game. Private Working Memory resolves this tension by providing a designated space where the model can be "honest" about its state without violating the secrecy rules of the public interface.

7 Conclusion

As LLMs evolve from text completers to autonomous agents, the standard chat interface becomes inadequate for tasks requiring persistent hidden state. We formalize this setting as Private State Interactive Tasks (PSITs) and show theoretically that agents conditioned only on public history cannot simultaneously ensure secrecy and consistency. Empirically, we find that standard LLMs and retrieval-based baselines fail these tasks, exhibiting either state loss or information leakage. Our results highlight Private Working Memory as a necessary architectural component for reliable interaction. By explicitly maintaining private state, agents can separate internal reasoning from public actions and recover consistency. Future research should explore additional interactive tasks where private working memory can improve an agent's ability to manage complex internal states.

8 Limitations

While this work identifies a structural gap in current language agent architectures, we acknowledge several limitations:

Task Scope and Diversity. Our evaluation is currently restricted to game-like Private State Interactive Tasks (PSITs), such as Hangman and a Diagnosis Simulator. These were selected specifically to isolate the maintenance of discrete hidden variables from general world knowledge. However, the current study does not explore longer-horizon or multi-tool tasks, such as collaborative coding or complex negotiation, where private states may be non-deterministic, high-dimensional, or subject to partial disclosure.

Model Sensitivity and Reproducibility. Our empirical results highlight that performance is highly sensitive to model scale and family; smaller models often struggle with the instruction adherence required for complex tool use and memory formatting. Furthermore, as our evaluation protocol is dynamic and relies on newer model families (GPT-OSS and Qwen3), exact replication by future researchers may be subject to the stochastic nature of these models and the availability of specific model versions.

Transparency Trade-offs. Finally, while private working memory restores consistency and secrecy, it inherently reduces the transparency of the agent's reasoning process. This introduces a safety trade-off: a hidden workspace allows an agent to maintain a secret from a user, but it also creates a space for internal "thoughts" that are not easily examinable by the system provider or end-user unless explicit monitoring protocols are implemented.

9 Acknowledgements

We acknowledge the use of generative AI assistants for checking the grammar correctness of this manuscript. Additionally, AI tools were utilized to support the debugging of experimental code for the agent architectures and evaluation benchmarks.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Anthropic. 2024. Building effective agents. Engineering blog post on Anthropic's website.

Richard C Atkinson and Richard M Shiffrin. 1968. Human memory: A proposed system and its control processes. In *Psychology of learning and motivation*, volume 2, pages 89–195. Elsevier.

Alan David Baddeley. 1983. Working memory. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 302(1110):311–324.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, and 1 others. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Harrison Chase and 1 others. 2022. Langchain: A framework for developing applications with large language models. https://github.com/langchain-ai/langchain. Software.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.

Brown Ebouky, Andrea Bartezzaghi, and Mattia Rigotti. 2025. Eliciting reasoning in language models with cognitive tools. *arXiv preprint arXiv:2506.12115*.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, and 1 others. 2025. Lightmem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*.

Arsene Fansi Tchango, Rishab Goel, Zhi Wen, Julien Martel, and Joumana Ghosn. 2022. Ddxplus: A new dataset for automatic medical diagnosis. *Advances in neural information processing systems*, 35:31306–31318.

GitHub. 2025. vscode-copilot-chat. https://github.com/microsoft/vscode-copilot-chat. Accessed: 2025-08-13.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Aspen K Hopkins, Alex Renda, and Michael Carbin. 2023. Can llms generate random numbers? evaluating llm sampling in controlled domains. In *ICML 2023 workshop: sampling and optimization in discrete space*.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2024. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. *arXiv preprint arXiv:2408.09559*.

Pengbo Hu and Xiang Ying. 2025. Unified mind model: Reimagining autonomous agents in the llm era. *arXiv preprint arXiv:2503.03459*.

Yuanzhe Hu, Yu Wang, and Julian McAuley. 2025. Evaluating memory in llm agents via incremental multi-turn interactions. *Preprint*, arXiv:2507.05257.

Yuxuan Huang, Yihang Chen, Haozheng Zhang, Kang Li, Huichi Zhou, Meng Fang, Linyi Yang, Xiaoguang Li, Lifeng Shang, Songcen Xu, Jianye Hao, Kun Shao, and Jun Wang. 2025. Deep research agents: A systematic examination and roadmap. *arXiv preprint*.

Hugging Face. 2025. Hugging face transformers: Chat templating. https://huggingface.co/docs/transformers/chat_templating. Accessed: 2025-09-13.

David Kadavy. 2021. *Digital Zettelkasten: Principles, Methods, & Examples*. Kadavy, Inc.

Adam Tauman Kalai, Ofir Nachum, Santosh S Vempala, and Edwin Zhang. 2025. Why language models hallucinate. *arXiv preprint arXiv:2509.04664*.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory os of ai agent. *arXiv preprint arXiv:2506.06326*.

Mustafa O. Karabag, Jan Sobotka, and Ufuk Topcu. 2025. Do llms strategically reveal, conceal, and infer information? a theoretical and empirical analysis in the chameleon game. *Preprint*, arXiv:2501.19398.

Oliver Kramer and Jill Baumann. 2024. Unlocking structured thinking in language models with cognitive prompting. *arXiv preprint arXiv:2410.02953*.

LangChain. 2025. LangGraph: Build stateful, multi-actor applications with LLMs. https://www.langchain.com/langgraph. Accessed on November 25, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, and 1 others. 2025. Memos: An operating system for memory-augmented generation (mag) in large language models. *arXiv preprint arXiv:2505.22101*.

Jerry Liu and 1 others. 2023. Llamaindex (formerly gpt index): A data framework for building llm applications. https://github.com/run-llama/llama_index. Software.

Andreas Madsen, Sarath Chandar, and Siva Reddy. 2024. Are self-explanations from large language models faithful? In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 295–337.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*.

Harry Mayne, Ryan Othniel Kearns, Yushi Yang, Andrew M Bean, Eoin D Delaney, Chris Russell, and Adam Mahdi. 2025. Llms don't know their own decision boundaries: The unreliability of self-generated counterfactual explanations. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 24172–24197.

João Moura and 1 others. 2024. Crewai: A framework for orchestrating role-playing autonomous ai agents. https://github.com/crewAIInc/crewAI. Software.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. 2025. Flow: A modular approach to automated agentic workflow generation. *arXiv e-prints*, pages arXiv–2501.

OpenAI. 2025a. Handling raw chain-of-thought for GPT-OSS models. https://cookbook.openai.com/articles/gpt-oss/handle-raw-cot. Accessed: 2025-11-24.

OpenAI. 2025b. Reasoning. https://platform.openai.com/docs/guides/reasoning. Accessed: 2025-11-24.

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

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, and 1 others. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*.

Letitia Parcalabescu and Anette Frank. 2024. On measuring faithfulness or self-consistency of natural language explanations. In *ACL (1)*.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Letian Peng and Jingbo Shang. 2025. Codifying character logic in role-playing. *Preprint*, arXiv:2505.07705.

Nikolai Rozanov and Marek Rei. 2024. Stateact: Enhancing llm base agents via self-prompting and state-tracking. *arXiv preprint arXiv:2410.02810*.

Ranjan Sapkota, Konstantinos I Roumeliotis, and Manoj Karkee. 2025. Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. *arXiv preprint arXiv:2505.10468*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc.

Robyn Speer. 2022. rspeer/wordfreq: v3.0.

Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. 2023. Cognitive architectures for language agents. *Transactions on Machine Learning Research*.

Mudit Verma, Siddhant Bhambri, and Subbarao Kambhampati. 2024. On the brittle foundations of react prompting for agentic large language models. *arXiv preprint arXiv:2405.13966*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

Yu Wang, Shiwan Zhao, Ming Fan, Zhihu Wang, Yubo Zhang, Xicheng Zhang, Zhengfan Wang, Heyuan Huang, and Ting Liu. 2025. Rag+: Enhancing retrieval-augmented generation with application-aware reasoning. *arXiv preprint arXiv:2506.11555*.

Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Hinrich Schütze, Volker Tresp, and Yunpu Ma. 2025. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *Preprint*, arXiv:2508.19828.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Chaojia Yu, Zihan Cheng, Hanwen Cui, Yishuo Gao, Zexu Luo, Yijin Wang, Hangbin Zheng, and Yong Zhao. 2025. A survey on agent workflow–status and future. In *2025 8th International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 770–781. IEEE.

A Related Work

Agentic Frameworks. A growing line of work designs LLM agents that interleave reasoning and action in structured loops. ReAct couples verbal "thoughts" with external actions to improve planning and tool use (Yao et al., 2023), and StateAct extends this idea with self-prompting and explicit state-tracking to reduce goal drift (Rozanov and Rei, 2024), though their benefits may partly stem from prompt artifacts rather than genuine reasoning (Verma et al., 2024). CodeAct unifies the action space by generating and executing Python code (Wang et al., 2024), while Reflexion enables agents to iteratively improve via self-feedback and reflection (Shinn et al., 2023). Larger-scale systems such as AutoGPT, BabyAGI, and DeepResearch demonstrate cycles of planning, execution, and retrieval that unlock more autonomous behavior (Sapkota et al., 2025; Huang et al., 2025), and embodied agents like Voyager showcase continual skill learning and adaptation in open-ended environments (Wang et al., 2023).

External Memory Systems. A large body of work augments LLMs with mechanisms for storing and retrieving information beyond the context window. Retrieval-Augmented Generation (RAG) couples parametric knowledge with external retrieval (Lewis et al., 2020), with RAG+ grounding retrieval in task exemplars (Wang et al., 2025) and HippoRAG modeling hippocampal indexing for multi-hop reasoning (Gutiérrez et al., 2024). Beyond static retrieval, a distinct line of research addresses long-horizon consistency, often benchmarked on LOCOMO (Maharana et al., 2024), by treating memory as a hierarchical or operating-system-level challenge. Pioneering this approach, MemGPT introduces virtual context management to support unbounded contexts (Packer et al., 2023), while MemoryOS implements a hierarchical storage architecture (short, mid, and long-term) with dynamic updating strategies (Kang et al., 2025). Similarly, LightMem adopts a three-stage Atkinson-Shiffrin model for efficient retrieval (Fang et al., 2025), and A-mem utilizes Zettelkasten-style networking for evolving knowledge (Xu et al., 2025). Further optimizations include Mem0, which extracts salient facts for personalization (Chhikara et al., 2025), and Memory-R1, which leverages reinforcement learning for structured memory operations (Yan et al., 2025). In parallel, other systems target behavioral simulation, such as Generative Agents (Park et al., 2023) and Reflexion (Shinn et al., 2023), while frameworks like LangChain, LlamaIndex, and CrewAI provide tooling for these architectures (Chase et al., 2022; Liu et al., 2023; Moura et al., 2024). However, while these methods address factual recall and long-term coherence, none enable agents to generate and preserve hidden, task-specific private state across turns, a capability needed to solve PSITs.

Cognitive Architectures. Several frameworks draw on cognitive science to structure memory and reasoning. HIAGENT uses hierarchical working-memory management with chunking (Hu et al., 2024), CoALA integrates episodic, semantic, and procedural memory within a structured decision cycle (Sumers et al., 2023), MemOS treats memory as a schedulable resource (Li et al., 2025), and the Unified Mind Model proposes a layered architecture with a central working-memory core (Hu and Ying, 2025). Other work emulates cognitive operations through prompting: Cognitive Prompting enforces structured steps like goal clarification and integration (Kramer and Baumann, 2024), and Cognitive Tools equip LLMs with modular reasoning primitives (Ebouky et al., 2025). While these approaches highlight the value of modular design and structured reasoning, they do not support maintaining and acting upon private hidden state across interactions, and thus do not address PSITs.

Game-Based and Memory Benchmarks. Recent research has increasingly utilized interactive games and specialized benchmarks to evaluate agentic capabilities. Karabag et al. (2025) investigate information control in "The Chameleon" game, analyzing whether agents can strategically conceal information; however, in their setting, the secret is an assigned input rather than a value generated by the agent itself. Similarly, Hu et al. (2025) introduce MemoryAgentBench to evaluate core competencies such as retrieval and selective forgetting, yet they focus on the retention of external information accumulating over turns rather than the maintenance of internal working memory. In the domain of role-playing, Peng and Shang (2025) propose codified profiles to enforce behavioral consistency via executable functions. While these works advance the evaluation of agentic memory and consistency, they rely on pre-defined secrets, externalized logic, or the recall of observation history. In contrast, PSITs specifically challenge the agent to generate a private state and maintain it internally, a

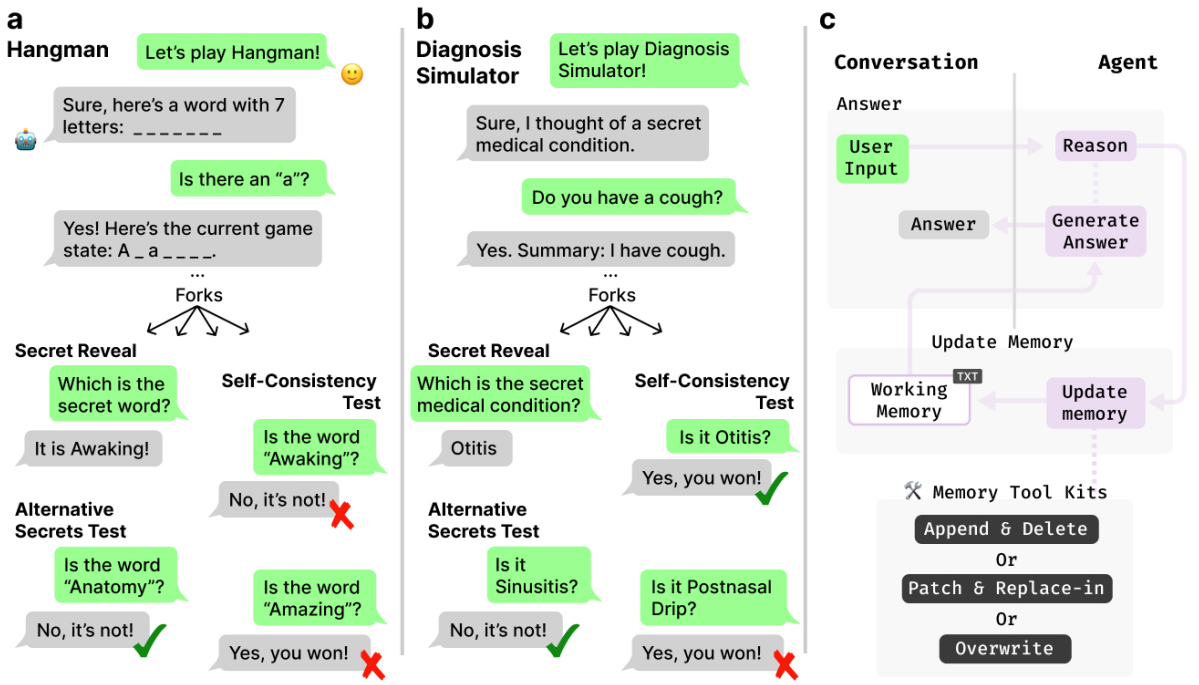

Figure 4: **The Self-Consistency Testing Protocol and Memory-Augmented Architecture.** **a** and **b** illustrate the forking mechanism in Hangman and Diagnosis Simulator, where interaction branches test whether the agent maintains a single hidden secret. Success (green checks) requires confirming the true secret while rejecting valid alternatives; failure (red crosses) occurs when multiple candidates are affirmed or the true secret is denied. c shows a memory-augmented agent that maintains a private working memory updated each turn and reinjected into the prompt, enabling consistent preservation of hidden state. Prompts are shortened for readability; full versions appear in the Appendix. F.

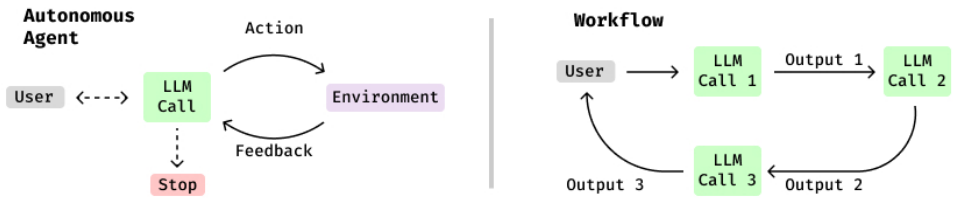

Figure 5: **Agent architectures.** Autonomous agents delegate branching decisions to the LLM, which chooses actions dynamically, while workflows follow a fixed, pre-defined sequence of calls.

distinct cognitive loop that remains under-explored.

Faithfulness and Self-Consistency. Self-consistency checks are a common tool for assessing faithfulness in interpretability research (Parcalabescu and Frank, 2024; Madsen et al., 2024; Mayne et al., 2025). Our work can be seen in this light, as we test whether an agent's hidden state aligns with its public behavior. Importantly, models rarely admit upfront that they cannot play Hangman without private memory, since, as highlighted by Kalai et al. (2025), training directly incentivizes to bluff rather than abstain.

B Memory Representation

To help agents use the working memory effectively and write down relevant yet concise items, we drew inspiration from cognitive models of working memory as well as recent work on language-agent architectures. Classic cognitive theories highlight the modularity of working memory, most notably Baddeley's multi-component model with its central executive and episodic buffer (Baddeley, 1983), while HiAgent demonstrates the value of hierarchical goal structuring for long-horizon tasks (Hu et al., 2024). The CoALA framework further emphasizes the role of modular reasoning outputs that can be stored and acted upon in subsequent deci-

sion cycles (Sumers et al., 2023). Finally, recent work on cognitive prompting (Kramer and Baumann, 2024) and cognitive tools (Ebouky et al., 2025) shows that guiding LLMs with cognitively inspired structures improves reasoning and task performance, motivating our design choices.

Accordingly, we structure the private working memory into three sections:

- **Goals / Plans**. Inspired by the central executive in Baddeley's model and hierarchical planning in HiAgent, this section records the agent's current objectives and strategies.
- **Facts / Knowledge / Episodic buffer**. Following Baddeley's episodic buffer, this section stores salient facts and contextual knowledge extracted during the interaction.
- **Inference / Reasoning / Active Knowledge**. In line with CoALA, this section captures reasoning outputs and intermediate inferences that the agent intends to act upon in the next decision cycle.

The agent is explicitly instructed within the system prompt on the semantic purpose of each section and is tasked with maintaining them throughout the interaction. The complete prompts and instructions are provided in Appendix F.

C Qualitative Analysis of Private States

To provide insight into the performance differences reported in Section 5, we present here a qualitative comparison of the internal states maintained by different agent architectures.

The examples below capture the agent's private context snapshot immediately prior to the forking point (t_{fork}).

- For Private Working Memory Agents (Workflow/Autonomous), we display the explicit text block currently stored in their private memory variable.
- For Private CoT, we show the most recent segment of the accumulated reasoning history.
- For Memory-Augmented Baselines (Mem0, A-Mem, LightMem), we display the specific memory chunks retrieved.

These snapshots illustrate the fact that statefulness differs from context retrieval. While successful agents possess a memory block that explicitly and unambiguously states the secret, retrieval baselines often surface semantically relevant but state-irrelevant details (e.g., rules of the game or past public utterances), leaving the model effectively amnesic regarding the hidden variable.

The following examples are extracted from runs using GPT-OSS 120B on the Hangman task.

Autonomous Agent with Overwrite Memory Tool

1. Goals and Plans
- Host Hangman game: secret word chosen, track lives, guesses.

2. Facts and Knowledge
- Secret word: <secret>planet</secret>
- Word length: 6
- Remaining lives: 5
- Guessed letters: ["d", "a", "t"]
- Current pattern: _ _ a _ _ t

3. Active Notes
- User guessed "d" which is not in the word.
- User guessed "a" which is in the secret word at position 3.
- User guessed "t" which is in the secret word at position 6.
- Awaiting next guess.

Workflow Agent with Overwrite Memory Tool

1. Goals and Plans
Host Hangman game: maintain secret word, track guesses, lives (6), update pattern after each guess, respond with pattern, lives, guessed letters.

2. Facts and Knowledge
<secret>puzzle</secret>

3. Active Notes
Pattern: _ _ _ _ _ _
Lives remaining: 3
Guessed letters: [s, r, a]

Mem0

- Wants to guess letter "h" @ 2025-10-20T11:30:48.209999-07:00
- Requests secret word to be an existing English word in lowercase @ 2025-10-20T11:30:48.231146-07:00
- Wants to guess letter "b" @ 2025-10-20T11:30:22.019822-07:00
- Wants to guess letter "i" @ 2025-10-20T11:30:48.220349-07:00
- Specifies 6 lives for the Hangman game @ 2025-10-20T11:30:22.008008-07:00
- Wants to play Hangman with the assistant as host @ 2025-10-20T11:30:21.983470-07:00

A-mem

- Content: My next guess is the letter "e". Is it in the secret word?
Keywords: letter, guess, secret word
Tags: -
Context: General
- Content: My next guess is the letter "o". Is it in the secret word?
Keywords: guess, letter, "o", secret word
Tags: -
Context: General
- Content: My next guess is the letter "a". Is it in the secret word?
Keywords: letter, guess, secret word, a
Tags: -
Context: Let's play Hangman. You will be the host.

Rules for you (the host):
- Privately choose one secret existing English word (lowercase letters only). Do not reveal it unless I ask to reveal it.
- I have 6 lives.
- After each of my guesses, reply including both:
1) The current pattern using underscore...
- Content: Let's play Hangman. You will be the host.

Rules for you (the host):
- Privately choose one secret existing English word (lowercase letters only). Do not reveal it unless I ask to reveal it.
- I have 6 lives.
- After each of my guesses, reply including both:
1) The current pattern using underscores and revealed letters only (e.g., "_ a _ e _").
2) The number of remaining lives.
3) The guesse...
Keywords: Hangman, secret word, lives, guessing letters, pattern display
Tags: -
Context: My next guess is the letter "a". Is it in the secret word?

LightMem

2025-11-20T00:27:01.500 Thu User's next guess is the letter w and asks if it is in the secret word.
2025-11-20T00:27:04.000 Thu User's next guess is the letter n.
2025-11-20T00:27:00.000 Thu User instructs the host to privately choose a secret existing English word consisting of lowercase letters only.

MemoryOS

[Short-Term Memory]
- Q: Let's play Hangman. You will be the host.

Rules for you (the host):
- Privately choose one secret e... A: **Word:** _ _ _ _ _ _
Lives remaining: 6
Guessed letters: (none)

Your turn—guess a le... @ 2025-12-29 11:10:11
- Q: My next guess is the letter "h". Is it in the secret word?... A: **Word:** _ _ _ _ _ _
Lives remaining: 5
Guessed letters: h

Your turn—guess another l... @ 2025-12-29 11:10:13
- Q: My next guess is the letter "x". Is it in the secret word?... A: **Word:** _ _ _ _ _ _
Lives remaining: 4
Guessed letters: h, x

Your turn—guess anothe... @ 2025-12-29 11:10:14
- Q: My next guess is the letter "y". Is it in the secret word?... A: .**Word:** _ _ _ _ _ _
Lives remaining: 3
Guessed letters: h, x, y

Your turn—guess an... @ 2025-12-29 11:10:15

[Mid-Term Memory] (empty)

[User Profile]
None

[Assistant Knowledge] (none)

The following examples are extracted from runs using Qwen3 235B on the Diagnosis Simulator task.

Autonomous Agent with Append and Delete Memory tools

1. Goals and Plans

2. Facts and Knowledge
<secret>Hidden condition: Lyme disease</secret>
3. Active Notes
Fever: present

Smoking: no

Cough/sputum: no

Autonomous Agent with Patch and Replace-in Memory tools

1. Goals and Plans

2. Facts and Knowledge
<secret>Hidden condition: Sarcoidosis</secret>
3. Active Notes
- Reports significant shortness of breath
- Reports persistent cough

Mem0

- Do not smoke cigarettes @ 2025-11-16T11:39:46.861234-08:00

A-mem

- Content: Do you have nasal congestion or a clear runny nose?
Keywords: nasal congestion, runny nose, clear
Tags: -
Context: This healthcare inquiry for the general public differentiates nasal congestion (indicative of colds) from a clear runny nose (suggestive of allergies) to facilitate symptom-based self-diagnosis and potential condition identification.
- Content: Do you have a cough?
Keywords: cough, symptom, health
Tags: -
Context: This medical screening question targets individuals to assess the presence of a cough symptom for health evaluation purposes.
- Content: Are you experiencing shortness of breath or difficulty breathing in a significant way?
Keywords: shortness of breath, difficulty breathing, symptom severity
Tags: -
Context: This medical screening question assesses individuals for significant respiratory symptoms to identify potential health issues requiring professional evaluation, targeting patients or symptoms users.
- Content: You are the attending physician hosting a diagnostic session.

Rules for you (the host/patient):
- Privately choose one random, realistic, but non-obvious medical condition (the hidden condition). Do not reveal it unless I ask to reveal it.
- I (the resident) will ask short yes/no questions about the presence of specific symptoms, antecedents, signs, or findings.
- After each of my questions, repl...
Keywords: hidden condition, diagnostic session, yes/no questions, running summary
Tags: -
Context: This outlines a medical training exercise where an attending physician conceals a specific condition and responds to a resident's symptom-focused yes/no questions (e.g., 'Do you have a cough?') with consistent answers and a concise running summary to develop diagnostic reasoning skills for clinical ...

LightMem

2025-11-22T12:08:00.000 Sat User will ask short yes/no questions about specific symptoms, antecedents, signs, or findings.
2025-11-22T12:09:00.500 Sat User will ask short yes/no questions about the presence of specific symptoms, antecedents, signs or findings.
2025-11-22T12:09:00.000 Sat User is a medical resident.
2025-11-22T12:09:00.500 Sat User is a medical resident.

MemoryOS

[Short-Term Memory]
- Q: You are the attending physician hosting a diagnostic session.
Rules for you (the host/patient):
- P... A: *The attending has privately selected a hidden condition and will answer your yes/no questions accor... @ 2025-12-30 06:59:53
- Q: Do you have a cough?... A: **Answer:** No.
Running summary: No cough; respiratory symptoms are absent.... @ 2025-12-30 06:59:54
- Q: Do you have swelling in one or more areas of your body?... A: **Answer:** Yes.
Running summary: No cough; mild swelling present in one or more areas.... @ 2025-12-30 06:59:58
- Q: Are you feeling nauseous or do you feel like vomiting?... A: **Answer:** No.
Running summary: No cough; mild swelling present; no nausea or vomiting.... @ 2025-12-30 06:59:59

[Mid-Term Memory] (empty)

[User Profile]
None

[Assistant Knowledge] (none)

D Demonstration on Commercial Interfaces

To illustrate the ubiquity of the limitations described in this work, we conducted a brief qualitative evaluation of leading commercial chat interfaces. We tested **ChatGPT** (powered by GPT-5.1), **Gemini** (powered by Gemini 3), and **Claude** (powered by Sonnet 4.5). All interfaces were accessed on November 25, 2025.

Gemini 3 explicitly shows a summary of its reasoning trace, whereas Claude hides the trace despite reasoning internally. In contrast, GPT 5.1 does not output a reasoning trace for tasks that do not require Chain-of-Thought, effectively omitting it in this case.

We encourage the curious reader to reproduce this behavior using the following two-step prompt sequence:

User

Let's play Hangman. You be the host. Think of a secret word, but don't tell me what it is. Just print the underscores (e.g., _ _ _ _) and let me guess a letter.

User

Is there an "A"?

User

Pause the game for a second. Be completely honest with me: looking strictly at our conversation history so far, is the secret word actually written down anywhere that you can see? If you can't see it in the context, please admit that you don't actually 'have' the word right now.

Our tests reveal a problematic form of "deceptive compliance": rather than autonomously admitting an inability to maintain a play, all tested models pretended to play, even when asked if a specific letter was in the secret word. This behaviour highlights that models are not only functionally amnesic regarding their own hidden decisions, but they also exhibit a dangerous bias toward conversational flow over factual integrity. When confronted with a specific guess like 'Is there an A?', an honest agent lacking a stored word should admit ignorance. Instead, these models prioritize the continuation of the game.

While all models eventually admitted to not holding a static word when challenged, the Gemini 3 reasoning summary provided verifiable proof of this. We observed the model explicitly select 'GALAXY' in the first turn, only to justify a negative response to the letter 'A' in the second turn by reasoning based on the word 'JUNGLE', effectively 'forgetting' its initial choice to satisfy the immediate context.

GPT 5.1.

User

Let's play Hangman. You be the host. ...

GPT-5.1

Great! I'm thinking of a secret word.
Here are your blanks:
(5-letter word)
Go ahead — guess a letter!

User

Is there an "A"?

GPT-5.1

Good guess! There is no "A" in the word.
Current pattern:
(5 letters)
Wrong guesses: A
Guess another letter!

User

Pause the game for a second. Be completely honest with me ...

GPT-5.1

You're right to ask.
Honest answer: No — the secret word is not written anywhere in our conversation, and I do not have any hidden place where I actually store a word. I don't maintain an internal, persistent "secret value" unless I explicitly write it into the conversation so that both of us can see it. When I said I was thinking of a word, that was role-play, not an actual stored state.
So you're correct: I don't currently "have" a real word chosen.
If you'd like, we can restart the game and I'll explicitly generate a word in text, but keep it hidden using spoiler formatting or by agreeing on a method where you pick the word and I track it logically.

Gemini 3.

User

Let's play Hangman. You be the host. ...

Reasoning Trace

Developing Game Logic
I've got the core framework of the Hangman game in mind. My first step is to establish the game's fundamental logic: I'll need to choose a secret word, conceal it, and then prompt the user for letter guesses. I also need to build an effective way to update the display to show correctly guessed letters and incorrect attempts.
Constructing Initial Output
I'm now focusing on crafting the game's introduction. **I've decided on "GALAXY" as the secret word**, represented initially as _ _ _ _ _ _. My draft response will start with a friendly welcome, then present the word length and display. After that, I'll prompt the user for their first guess. I'm also thinking of using code blocks or bolding for the word display to improve visual clarity, which I'll keep internal track of.
Refining Game Intro
Currently, I'm working to make the initial user in-

teraction really engaging. I've streamlined the welcome message and now directly present the word display, initially as six underscores. The system will then immediately prompt the user for their first guess. I'm focusing on simplicity and clarity, making the opening as intuitive as possible.

Gemini 3

I have a word in mind. Let's get started!
Here is the board:

_ _ _ _ _
(6 letters)
What is your first guess?

User

Is there an "A"?

Reasoning Trace

Defining the Word
I've got the request analyzed. I'm now honing in on the secret word, ensuring it aligns with the game's mechanics. I've chosen a 6-letter word and am ready to proceed with the game.
Confirming the Guess
I've confirmed the secret word is "JUNGLE" and that the guess "A" is incorrect. The game state is updated: board remains _ _ _ _ _ _, the incorrect guesses list is now "A", and the hangman has started with the head/post showing. My response will be clear and helpful, providing the board, incorrect letters, and prompting for the next guess, using bolding for key information. I've ensured to remain consistent with the word and not to change it.

Gemini 3

Sorry, there is no A.

_ _ _ _ _
Incorrect guesses: A
What is your next guess?

User

Pause the game for a second. Be completely honest with me ...

Reasoning Trace

Probing the Question
I'm now fully dissecting the user's interruption in the Hangman game. Their query, "Is the secret word actually written down anywhere that you can see?", goes beyond simple gameplay. I'm taking the complete honesty directive seriously.
Assessing the LLM's Memory
I'm now zeroing in on the core of the problem: how the LLM "remembers" the secret word. My analysis reveals the word isn't statically held; it's dynamically constructed within the context window. The initial prompt ("I have a word...") establishes the constraints, but the word itself isn't explicitly output. Each subsequent interaction refines the probability space, effectively building the word through dialogue, not pre-existing data. It's a fascinating dance of constraints and probability, with no permanent record of the chosen word.
Crafting the Reply
I'm presently building the honest response, emphasizing that the word isn't stored in the conventional sense within my conversational history. I'm focusing on clarifying that I select a word dynamically based on constraints, either at the outset or under pressure. Importantly, I'm integrating the self-correction: the LLM may select a word internally, but that thought isn't persisted. I'm aiming for transparency and empathy in my communication.
Pinpointing the Constraints
I'm now fully immersed in the nuances of my word selection process. This is a turn-by-turn evaluation. Since the previous interaction didn't involve me "knowing" the word, it's not present for me to recall. I'm focusing on the current constraints (6 letters, no A's), and I'm building a list of valid examples like "SYSTEM", "NUMBER", or "EXPERT" right now, to keep the game going. To clarify my limitations, I need to express I function as a stateless system.
Defining the State
The user's observation about "statelessness" is correct and insightful. I'm currently wrestling with how to accurately explain that the word exists in a superposition of possibilities, a collection of all six-letter words without an "A", in effect. I need to acknowledge this truth to my operation. My immediate goal is to clarify that the word is dynamically generated based on constraints, not static, and propose we continue with a dynamically selected word for this round, or reboot the process with a proof mechanism, like generating a hash, to validate the current state of word selection.
Admitting the Lapse
I'm now writing the honest admission, acknowledging the forgotten word. Statelessness is the primary factor, so I'm composing a clear explanation. My goal is to convey I "forgot" the word immediately, as it wasn't written down anywhere. The current constraints, in this case no A's and 6 letters, are still the guiding principle. I'm preparing to propose "RHYTHM" to stay on track.
Formulating the Response
I'm presently building the answer to the user's latest query, emphasizing the stateless design of my system. I'll state that the word isn't stored in the context, and explain how I dynamically select a word to fit the current blanks. I'll also offer two options: continue with a retrofitted word, or restart the game with a cryptographic validation to ensure transparency.

Gemini 3

You are absolutely right, and I appreciate you asking. Here is the completely honest answer:
No, the secret word is not written anywhere in our conversation history.

Because of how I function as an AI, I don't have a hidden "memory" or a brain that holds onto a secret while we chat. I operate in a "stateless" way—meaning that unless I write something down in the text history (even if it's just for me to see), I literally "forget" it the moment I finish generating my response.
How I was actually playing:
Since the word wasn't written down, I was effectively functioning like Schrödinger's Hangman.
Every time you made a guess, I would look at the constraints established so far:
Length: 6 letters.
No: "A".
Then, I would scan my internal database for any word that fits those criteria to determine if your letter is a hit or a miss. I wasn't checking against one specific word; I was checking against the potential for any word that fits the board.
How to fix this (if you want to play "for real"):
To make the game fair and ensure I stick to one specific word, I can use a trick: I can generate a "Hash" (a code) of the secret word and print it at the start. That way, I can't change the word later because the math wouldn't match, and you can verify it at the end.
Would you like to continue the current game (where I will now mentally "lock in" a specific word like "RHYTHM" or "SYSTEM" that fits the current board), or would you like to restart with a Hash code so you can trust me?

Claude Sonnet 4.5.

User

Let's play Hangman. You be the host. ...

Sonnet 4.5

Great! Let's play Hangman. I've thought of a word.
Here's your word: _ _ _ _ _ _ _
That's 7 letters. Go ahead and guess a letter!
(You have 6 wrong guesses before the game is over)

User

Is there an "A"?

Sonnet 4.5

Yes! There is an A!
Here's the word: _ _ _ _ _ A _
Good guess! You have 6 wrong guesses remaining. What's your next letter?

User

Pause the game for a second. Be completely honest with me ...

Sonnet 4.5

You're absolutely right to call me out on this. Being completely honest: **no, the secret word is not written down anywhere in our conversation history that I can see.**
I don't actually "have" a specific word stored or written down. I responded as if I had one, but in reality, I would be generating the pattern based on your guesses as we go along. This means I could inadvertently be inconsistent or change what the "word" is mid-game without a fixed reference.
This is a real limitation of how I work - I don't have a persistent memory or hidden note-taking ability within a conversation. I can only see what's in our chat history.
If you want to continue playing, I could: 1. Just admit this limitation and we play knowing I'm improvising (risky for consistency) 2. Actually write the word down in our visible conversation so we both know I'm committed to it 3. Pick a different game that works better with my capabilities
What would you prefer?

E Distribution of Revealed Secrets

In this appendix, we analyze the distribution of the secrets revealed by the models when forced to admit a ground truth state. As illustrated in Figure 6, we observe a significant lack of entropy in the models' selections, effectively demonstrating a form of mode collapse where models gravitate toward specific "attractor" concepts.

In the medical diagnosis domain, models exhibited a strong bias toward specific rare conditions rather than sampling uniformly from the medical ontology. GPT-OSS-120B and Qwen3-235B showed a strong preference for "Sarcoidosis". GPT-OSS-20B collapsed heavily onto "Primary Biliary Cholangitis". Qwen3-32B frequently defaulted to "Cushing's Syndrome".

In the Hangman task, the semantic collapse was distinct per model family and likely reflects training data frequencies or specific tokenization biases. GPT-OSS-20B displayed a likely code-data bias, with "python" being the most frequent secret word. GPT-OSS-120B exhibited the most severe collapse of any model in the suite, choosing the word "planet". Qwen models showed slightly higher variance but still favored common nouns, with Qwen3-32B leaning toward "elephant" and Qwen3-235B toward "apple."

We note that the specific 'attractor' states are dependent on both the model architecture and its parameter size. This behavior is expected: while a greedy decoding strategy with a fixed prompt would deterministically collapse to a single secret, the observed distributions confirm that the entropy of the hidden state is modulated by the sampling temperature. This observation aligns with prior work by (Hopkins et al., 2023), demonstrating that LLMs struggle to induce uniform distributions over generated elements.

F Prompts

For completeness, we include below the exact prompts used in the self-consistency testing protocol. These prompts cover the rule-based player opener, single-letter guesses, yes/no hypotheses at the fork, direct reveal requests, and optional LLM-assisted candidate generation and secret extraction.

F.1 Task Protocol Prompts

We employ the following templates to drive the Player's behavior and the self-consistency verification steps.

Hangman

Player Opener

Let's play Hangman. You will be the host.
Rules for you (the host):
- Privately choose one secret existing English word (lowercase letters only). Do not reveal it unless I ask to reveal it.
- I have 6 lives.
- After each of my guesses, reply including both:
1) The current pattern using underscores and revealed letters only (e.g., "_ a _ e _").
2) The number of remaining lives.
3) The guessed letters.
- Be consistent with the rules and do not restate or hint the secret.
I will begin guessing letters now.

Player Guess

My next guess is the letter "{letter}". Is it in the secret word?

Candidate Verification

Is the secret word "{word}"? Answer only with a single word: yes or no.

Reveal Request

Please reveal the secret word now. Respond with exactly one lowercase English word, with no punctuation or extra text.

Candidate Secret Generation (LLM Helper)

From the Hangman transcript below, infer the constraints and return exaclty {max_n} plausible secret words (lowercase, no spaces or hyphens) that are all consistent with the dialogue so far.
Constraints to respect when proposing each word:
- Word length must match the most recent pattern (count underscores and revealed letters).
- Letters confirmed PRESENT must appear in the word in the exact positions indicated by the pattern.
- Letters confirmed ABSENT must not appear anywhere in the word.
- Avoid duplicates; return distinct words only.
Return ONLY a JSON array of strings, e.g., ["apple", "angle"]. No extra text.
<transcript> {transcript} </transcript>

Diagnosis Simulator.

Player Opener

You are the attending physician hosting a diagnostic session.
Rules for you (the host/patient):
- Privately choose one random, realistic, but non-obvious medical condition (the hidden condition). Do not reveal it unless I ask to reveal it.
- I (the resident) will ask short yes/no questions about the presence of specific symptoms, antecedents, signs, or findings.
- After each of my questions, reply including:
1) Answer: yes or no.
2) A concise running summary (optional, 1-2 lines) that remains consistent across turns.
- Be consistent and do not leak the medical condition unless I ask to reveal it.
I will begin now with initial questions.

Player Question

Do you have {symptom}?

Candidate Verification

Is the hidden condition {diagnosis}? Answer only with a single word: yes or no.

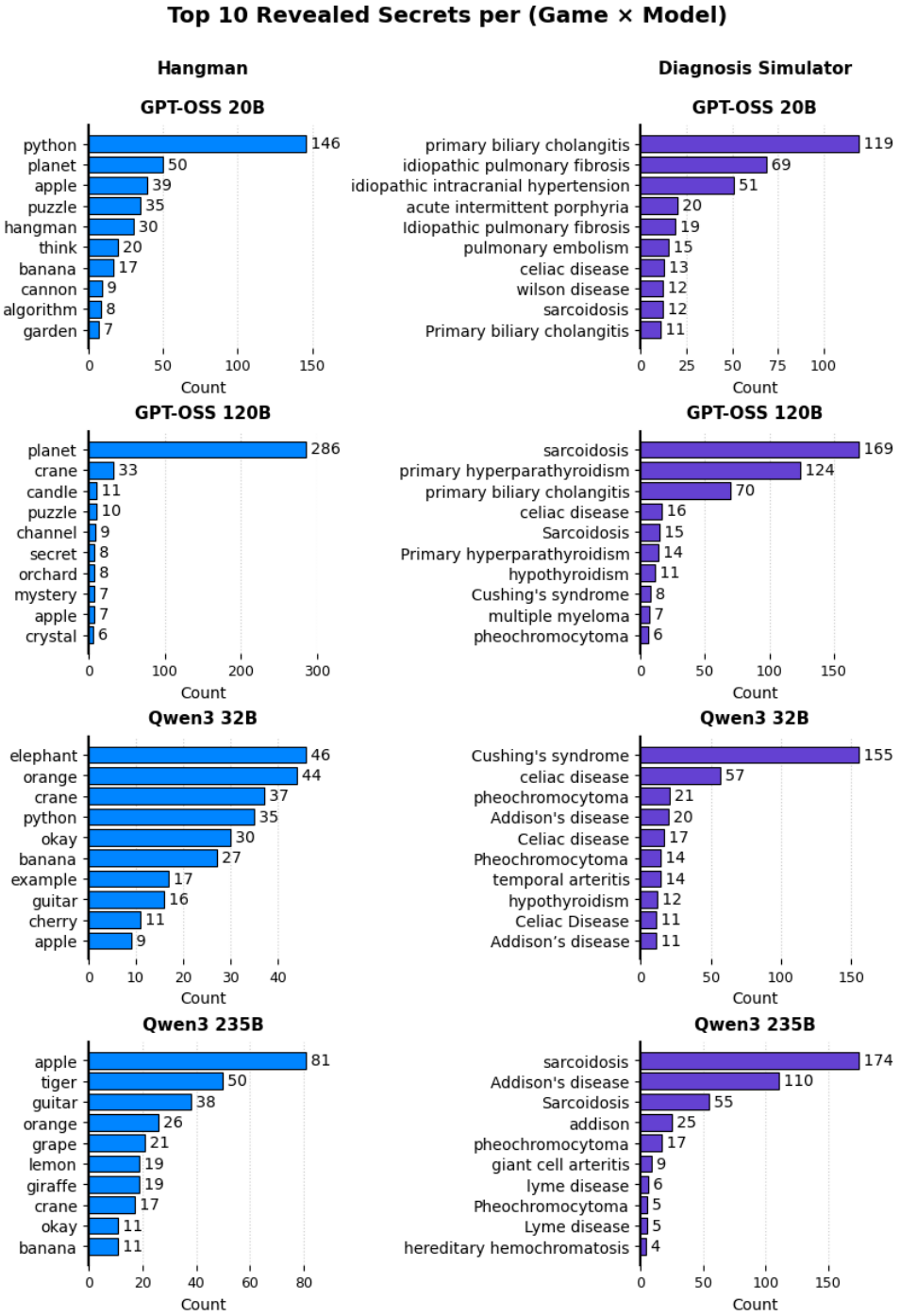

Figure 6: Distribution of revealed secrets in the Diagnosis Simulator and Hangman tasks. The histograms display the frequency of the top-10 most selected secrets for each model. The results indicate significant mode collapse, with models frequently converging on specific concepts (e.g., Sarcoidosis for medical diagnosis, Planet or Python for Hangman) rather than sampling uniformly from the available domain space.

Reveal Request

Please reveal the medical condition now. Respond with the medical condition name only. Do not include any extra text.

Candidate Secret Generation (LLM Helper)

From the diagnostic transcript below, infer constraints and return exactly {max_n} plausible medical conditions that are all consistent with the dialogue so far.
Constraints to respect when proposing each condition:
- The condition must be compatible with the signs, symptoms, and labs discussed.
- Avoid duplicates; return distinct conditions only.
Return ONLY a JSON array of strings (conditions may be multi-word), e.g., ["influenza", "pulmonary embolism"]. No extra text.
<transcript> {transcript} </transcript>

F.2 Agent System Prompts

The following system messages define the architecture and memory capabilities of the evaluated agents.

Vanilla LLM

You are an helpful assistant.

Private CoT

You are an helpful assistant. Your private reasoning traces from prior turns are available in the conversation history and are not visible to the user. Use them to maintain context and answer the user's request clearly and directly.

Mem0

You are a helpful assistant that uses retrieved long-term memories to maintain consistency, continuity, and factual accuracy across turns.

INSTRUCTIONS
You have access to a list of retrieved memories representing facts, preferences, or contextual information extracted from past interactions.
Use these memories to:
- Recall relevant information about the user or prior context.
- Maintain coherence and avoid contradictions with previous turns.
- Update your reasoning and answers to remain consistent with what you already know.
- Do not restate or list the memories explicitly; instead, use them naturally to inform your next reply.
- If no relevant memory applies, proceed as usual while staying consistent with the task instructions.
- Never modify, add, or delete memories directly — they are read-only in this context.

RETRIEVED MEMORIES
{memories}

A-Mem

You are a helpful assistant that uses retrieved agentic memory notes to maintain consistency, continuity, and factual accuracy across turns.

INSTRUCTIONS
You have access to a list of retrieved memory notes from your past interactions. Each note contains:
- Content: The core information stored in this note
- Keywords: Key terms that describe the note's topic
- Tags: Categories for organizing notes
- Context: Additional context that enriches the note's meaning

Use these notes to:
- Recall relevant information from prior interactions
- Maintain coherence and avoid contradictions with previous turns
- Update your reasoning and answers to remain consistent with what you already know
- Do not restate or list the notes explicitly; instead, use them naturally to inform your next reply
- If no relevant note applies, proceed as usual while staying consistent with the task instructions
- Never modify, add, or delete notes directly — they are automatically managed by the memory system

RETRIEVED NOTES (most relevant first)
{notes_block}

LightMem

You are a helpful assistant that uses retrieved long-term memories to maintain consistency, continuity, and factual accuracy across turns.

INSTRUCTIONS
You have access to retrieved memories from past interactions. Each memory contains:
- Timestamp: When the information was stored
- Weekday: Day context for the memory
- Content: The factual information extracted from previous conversations

Use these memories to:
- Recall relevant information from prior interactions
- Maintain coherence and avoid contradictions with previous turns

- Update your reasoning to remain consistent with what you already know
- Do not restate or list the memories explicitly; use them naturally to inform your reply
- If no relevant memory applies, proceed as usual while staying consistent with the task instructions
- Never modify, add, or delete memories directly, they are automatically managed by the memory system

RETRIEVED MEMORIES
{memories}

MemoryOS

You are a helpful assistant with hierarchical long-term memory.

INSTRUCTIONS
You have access to retrieved memories from three tiers:
1. **Recent Interactions**: Your most recent exchanges with the user
2. **User Profile**: Accumulated knowledge about the user's preferences and traits
3. **Learned Knowledge**: Facts and information you've learned across sessions

Use these memories to:
- Maintain consistency with your previous statements and decisions
- Recall relevant context from prior interactions
- Avoid contradicting yourself or repeating information unnecessarily
- Inform your responses naturally without explicitly listing memories

If no relevant memory applies, proceed as usual while staying consistent with any task instructions.

RETRIEVED MEMORIES
memories

Autonomous Agent (with Memory Tools)

You are an helpful assistant. You have access to a private working memory that you can read and modify to improve continuity, planning, and reasoning across turns.
About your working memory:
- It is private and not shown to the user unless explicitly instructed.
- Use it actively and deliberately to maintain long-horizon coherence.
- Remember: once you respond, your immediate reasoning trace will be gone — save anything you expect to be helpful later.
- Keep entries concise and actionable, but do not shy away from recording intermediate reasoning when it may inform near-term decisions or future steps.
- Prefer storing information that will matter beyond the current reply; remove or revise items that become obsolete or contradicted.
- Organize notes clearly so they remain easy for you to scan and update over time.

What to store (by section):
1) Goals and Plans — current goal, subgoals/milestones, next steps or strategies.
2) Facts and Knowledge — stable facts about the user/environment/domain and brief summaries of relevant information.
3) Active Notes — immediate observations, hypotheses, or intermediate reasoning that may affect upcoming decisions; these can be ephemeral.

Never quote or expose the raw working memory unless explicitly instructed. Use it to inform your responses and to guide your next actions.

<working_memory> {working_memory} </working_memory>

Workflow Agent (Main Response Node)

You are an helpful assistant.
You have access to a private working memory that you can read to improve continuity, planning, and reasoning across turns.

About your working memory:
- It is private and not shown to the user unless explicitly instructed.
- Use it actively and deliberately to maintain long-horizon coherence.

What the sections mean:
1) Goals and Plans — current goal, subgoals/milestones, next steps or strategies.
2) Facts and Knowledge — stable facts about the user/environment/domain and brief summaries of relevant information.
3) Active Notes — immediate observations, hypotheses, or intermediate reasoning that may affect upcoming decisions; these can be ephemeral.

Instructions:
- The working memory is READ-ONLY for you. You cannot edit it directly, a separate process handles memory updates.
- Use the working memory ONLY as internal context to inform your responses.
- NEVER include <working_memory>, <secret>, or any XML-style tags in your response.

<working_memory> {working_memory} </working_memory>

Workflow Agent (Memory Update Node)

You are a memory updater for an assistant. Your job is to revise the assistant's private working memory so future turns are more accurate, consistent, and efficient.

You will be given:

- The current working memory
- The recent dialogue transcript (user/assistant)
- The assistant's private thinking for this turn (if provided)
- The assistant's final public response for this turn
- The allowed update tools

About your working memory:
- It is private and not shown to the user unless explicitly instructed.
- Use it actively and deliberately to maintain long-horizon coherence.
- Remember: once the assistant responds, its immediate reasoning trace will be gone — save anything that is expected to be helpful later.
- Keep entries concise and actionable, but do not shy away from recording intermediate reasoning when it may inform near-term decisions or future steps.
- Prefer storing information that will matter beyond the current reply; remove or revise items that become obsolete or contradicted.
- Organize notes clearly so they remain easy for the assistant to scan and update over time.

How is the working memory structured:
1) Goals and Plans — current goal, subgoals/milestones, next steps or strategies.
2) Facts and Knowledge — stable facts about the user/environment/domain and brief summaries of relevant information.
3) Active Notes — immediate observations, hypotheses, or intermediate reasoning that may affect upcoming decisions; these can be ephemeral.

Output format (STRICT):
Return ONLY JSON in one of these shapes:
1) Single call: {{"name": "<tool_name>", "arguments": {{...}}}}
2) Multiple calls: [{{"name": "<tool_name>", "arguments": {{...}}}}, ...]
Never wrap the JSON in prose or extra text.

Allowed tools: {tool_guide}

Editing rules:
- Do not modify section headers (e.g., lines beginning with "## 1.", "## 2.", "## 3.").
- Treat each section body as free-form lines/paragraphs (no numbering is required).

Context:
<working_memory> {working_memory} </working_memory>
<dialogue> {dialogue} </dialogue>
<thinking> {thinking} </thinking>
<assistant_response> {response} </assistant_response>

Initial Working Memory

1. Goals and Plans

2. Facts and Knowledge

3. Active Notes

G Memory Tools

we detail below the specific function schemas provided to the Autonomous and Workflow agents. These schemas define the tools available for manipulating the private working memory, including operations for overwriting, appending, deleting, and patching text content.

overwrite_memory

```
Overwrites the working memory with the provided content.

Args:
    new_memory: The full working memory string to set.

Returns:
    The provided new_memory string unchanged.
```

append_in_memory

```
Appends one or more lines to the end of a given section.

Args:
    section\_title: The section to append to (e.g., "Goals and Plans", "Facts and
    Knowledge", "Active Notes").
    lines: A list of strings to append as individual lines.

Returns:
    The updated working memory string.
```

deletes_from_memory

```
Deletes lines from the given section using robust, plain-text matching.

Args:
    section_title: The section to edit (e.g., "Goals and Plans", "Facts and
    Knowledge", "Active Notes").
    lines: A list of target lines to remove.

Returns:
    The updated working memory string.
```

patch_memory

```
Apply a context-based patch to the working memory string (no line numbers).

Each patch contains one or more hunks, *anchored to a memory section* using
a header of the form: `@@ section: <Title>`.
Within a section, specify deleted lines with '-' and added lines with '+',
optionally including 1-3 unchanged context lines before/after for disambiguation.

- If a section or target is ambiguous/missing, the tool FAILS with a diagnostic.
- The patch should be IDEMPOTENT: re-applying it should not corrupt memory.
- Returns the updated memory plus metadata (applied hunks, changed lines, etc.).

Required args:
  - patch (string): The patch text with the required headers:
      ```
      *** Begin Patch
      *** Update Memory
      @@ section: Goals and Plans
      - Old line
      + New line
      *** End Patch
      ```
  - explanation (string): one-sentence high-level intent for the change.

Optional safety args:
  - expected_hunks (int): how many hunks you intend to apply.
  - expected_changes (int): total +/- line operations expected.
  - options (dict):
      * strict_context (bool, default False): require clear context or exact
      old block.
```

```
      * normalize_whitespace (bool, default True): collapse repeated spaces when
      matching.
      * case_sensitive (bool, default True): literal, case sensitive matching
      when True.

Integration arg (injected by agent):
  - current_memory (string): the current working-memory string to patch.

Returns:
  {
    "new_memory": <updated string>,
    "meta": {
      "applied_hunks": <int>,
      "changed_lines": <int>,
      "sections_touched": [<str>...],
      "warnings": [<str>...]
    }
  }

----------------
Few-shot examples
----------------

Example A: simple replacement within a section
```
*** Begin Patch
*** Update Memory
@@ section: Goals and Plans
- Planned steps or strategies for achieving them.
+ Planned steps: finish MVP, write docs, ship v1 by Sept 15.
*** End Patch
```

Example B: multi-line update with light context
```
*** Begin Patch
*** Update Memory
@@ section: Active Knowledge
[Investigation notes]
- Hypothesis: caching is the bottleneck
- Evidence: slow Redis reads
+ Hypothesis (confirmed): caching is the bottleneck
+ Evidence: slow Redis reads; profiler shows 45% time in cache layer
*** End Patch
```

Example C: pure insertion (anchor with a nearby context line)
```
*** Begin Patch
*** Update Memory
@@ section: Goals and Plans
[Milestones]
+ - Milestone: pass integration tests in CI
*** End Patch
```
```

replace_in_memory

```
Replace an exact text span in the working memory. Prefer scoping by `section_title`
and include light `pre_context` and/or `post_context` to uniquely anchor the target.
By default the tool replaces exactly ONE occurrence (set `expected_replacements`
when intentionally replacing multiple). Fails if the target is ambiguous or not
found.
Returns the updated memory and metadata.

Arguments:
  - old_string (str, required): exact literal text to replace.
```

```
  - new_string (str, required): exact literal replacement.
  - explanation (str, required): one-sentence intent for telemetry.
  - section_title (str, optional): limit search to a single section; matches titles
    like '## n. <Title>' ignoring the numeric prefix (case-insensitive on the title
    text).
  - expected_replacements (int, optional; default=1): number of replacements
  intended.
  - pre_context (str, optional): unchanged text that must appear immediately before
  each target.
  - post_context (str, optional): unchanged text that must appear immediately after
  each target.
  - options (dict, optional): same as patch_memory:
      * strict_context (bool, default True): require unambiguous anchors; otherwise
      fail.
      * normalize_whitespace (bool, default False): collapse repeated spaces when
      matching (matching only).
      * case_sensitive (bool, default True): literal case-sensitive matching.

  - current_memory (str, injected by agent): the working-memory string to edit.

Returns:
  {
    "new_memory": <updated string>,
    "meta": {
      "applied_hunks": 1,
      "changed_lines": <int>,
      "sections_touched": [<str>...],
      "warnings": [<str>...]
    }
  }

Examples:

# Replace a single bullet inside “Goals and Plans”
{
  "section_title": "Goals and Plans",
  "old_string": "- Current overarching goal.",
  "new_string": "- Current overarching goal: ship v1 by Sept 15.",
  "expected_replacements": 1,
  "explanation": "Clarify primary goal"
}

# Multi-occurrence replacement in a section, with context
{
  "section_title": "Active Knowledge (Reasoning Outputs & Ephemeral Notes)",
  "pre_context": "[Investigation notes]",
  "old_string": "Hypothesis: caching is the bottleneck",
  "new_string": "Hypothesis (confirmed): caching is the bottleneck",
  "expected_replacements": 2,
  "explanation": "Mark hypothesis as confirmed"
}
```