ARSM: AUTO-REGRESSIVE STATE MACHINE FOR AGENTIC REASONING COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

While Large Language Model (LLM)-based agents demonstrate strong capabilities in long-horizon tasks by interleaving reasoning with external environment interactions, the continuous accumulation of context rapidly creates a critical memory bottleneck. Existing memory compression methods rely on task-specific optimization or external auxiliary models, introducing significant computational overhead. Furthermore, the resulting compressed representations tend to lose structured relationships, leading to information dilution, attention collapse, and degraded decision consistency. To address these limitations, we propose **A**uto-**R**egressive **S**tate **M**achine (ARSM), a lightweight training-free framework that enables in-situ reasoning compression through structured state evolution. ARSM introduces two key components: (i) a trajectory abstraction mechanism that reorganizes interaction histories into compact **H**ypothesis–**A**ction–**R**esult (HAR) micro-chains; (ii) a dynamic state machine that regulates hierarchical memory through atomic operations and a compression-control parameter. These components are unified within an auto-regressive, self-compressive generation space, where each model output jointly performs external action execution and internal state updates. We evaluate ARSM on Webshop, Multi-Objective Multi-Hop QA, and SWE-Bench Lite datasets. Experimental results show that ARSM maintains the task performance while simultaneously reducing token consumption, offering a practical, cost-effective route toward scalable autonomous agents for long-horizon tasks.

1 INTRODUCTION

Current agentic artificial intelligence (Agentic AI) models have shown remarkable capabilities in solving complex tasks. Built upon Chain-of-Thought (CoT) reasoning (Wei et al., 2022) and tool-use mechanisms (Schick et al., 2023), these agents typically operate through a multi-turn reasoning–action–observation interaction paradigm (Yao et al., 2023). In complex domains such as software engineering (SWE), this interleaved reasoning methodology (Yang et al., 2024b) has proven effective in handling long-horizon tasks, including code debugging and error localization.

However, as interaction histories grow, agents must repeatedly process observations that differ substantially in their relevance to future decisions. In software debugging, for example, a file inspection or test run may produce a long output, whereas a later decision may depend on only the hypothesis being tested, the action taken, and the resulting evidence. Retaining every output increases context usage, while removing earlier interactions can discard information needed to interpret subsequent actions. This motivates a memory representation that distinguishes ongoing investigations, completed sub-tasks, and failed attempts, rather than treating all past interactions identically.

To address the challenges above, prior work has explored memory management and context compression strategies. Methods such as LLMLingua (Jiang et al., 2023) and retrieval augmentation (Fang et al., 2025) adopt token compression schemes and prune historical contexts. Recent approaches (Su et al., 2026; Ye et al., 2025) attempt to compress historical trajectories through learned policies, retrieval augmentation, or external distillation modules. However, it adds computational overhead and lacks agentic real-time intuition, risking technical detail loss during summarization (Luo et al., 2026).

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

These observations motivate a more specific question: how can an agent maintain a compact task state that supports subsequent decisions as its interaction history grows? We propose **A**uto-**R**egressive **S**tate **M**achine (ARSM), a training-free framework for structured context management. ARSM combines compact records of past interactions with explicit rules for updating task progress, failure lessons, milestones, and active reasoning paths. Its central objective is to reduce the amount of historical context carried forward while retaining information relevant to the ongoing task.

Specifically, ARSM consists of two core components. The first one is the **trajectory abstraction via Hypothesis–Action–Result (HAR) micro-chains**. We reorganize the conventional Reasoning—Action—Observation paradigm into Hypothesis–Action–Observation–Result chain, which can be abstracted into a logical HAR micro-chain. As this is executed by the model itself, it can intrinsically retain the reasoning focus and filter out redundant environmental noise while preserving the underlying structural causal topology of the reasoning process. The second one is the **dynamic state machine and sliding window**. Building upon abstraction, we introduce a dynamic state machine to represent and maintain a hierarchical internal state (e.g., task progress and failure constraints), evolved by a set of atomic operations (e.g., push, fold, prune). It also employs a dynamic sliding window mechanism to explicitly bound the token length with a compression-control parameter, allowing for a controllable trade-off between working memory capacity and state sparsity.

These two components are then unified within an auto-regressive, self-compressing generation space, where each model output yields: (i) executable actions for environmental interaction, and (ii) structured metadata for updating the internal state. As illustrated in Figure 1, this establishes a dual-loop architecture that tightly couples external execution with timely internal state evolution.

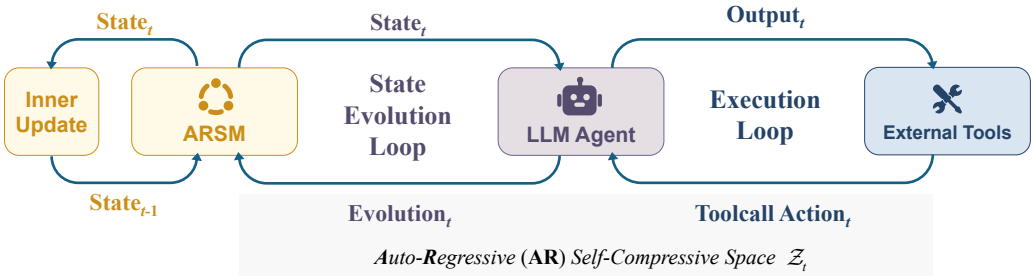

Figure 1: **Overview of the dual-loop architecture.** It contains two cyclical processes: (i) execution loop, which manages direct interactions with external tools and feeds back environmental output; and (ii) state evolution loop, which synchronizes agent's cognition with ARSM. The Auto-Regressive Self-Compressive Space acts as a unified generation space, dispatching tool-call action to the environment and evolution metadata to drive ARSM's inner state update: $\text{State}_{t-1} \rightarrow \text{State}_t$.

We conduct a systematic evaluation of the proposed method on three representative datasets: WebShop (Yao et al., 2022), Multi-Objective Multi-Hop QA (Yang et al., 2018; Ho et al., 2020; Trivedi et al., 2022; Aksitov et al., 2023), and SWE-Bench Lite (Jimenez et al., 2024). Experimental results show that on WebShop, ARSM consistently improves the reward across different backbone model families; on 16-Objective Multi-Hop QA, it achieves improved EM and F1 scores. Across all evaluated datasets, ARSM substantially reduces token costs by up to 82%. These findings demonstrate that structured chain construction and state compression are key factors in improving both efficiency and performance in long-horizon reasoning.

Our main contributions are summarized as follows:

- We introduce **A**uto-**R**egressive **S**tate **M**achine (ARSM), a training-free context-management framework in which the agent generates memory-update metadata alongside its next action, without requiring fine-tuning or an auxiliary compression model.
- We combine Hypothesis–Action–Result records with an event-driven controller that manages task progress, failure lessons, milestones, and active interaction records. A compression control parameter adjusts the capacities of structured memory and recent raw history.
- We evaluate ARSM on WebShop, Multi-Objective Multi-Hop QA, and SWE-bench Lite. The results characterize its task-dependent benefits and cost–accuracy trade-offs, while component comparisons and trajectory analyses examine the behavior and limitations of the proposed memory design.

2 Related Work

Interleaved Reasoning in Agents. Building upon Chain-of-Thought (CoT) reasoning (Wei et al., 2022) and tool-use integration (Schick et al., 2023; Chen et al., 2024), LLM-based agents employing interleaved reasoning paradigms, characterized by multi-turn interactions (Yao et al., 2023; Shinn et al., 2023), have demonstrated superior performance in solving complex problems. Recent work (Jin et al., 2025; Chen et al., 2024) has further improved general-purpose agentic capabilities through task-specific adaptation. In specialized domains such as software engineering (SWE), frameworks such as SWE-Agent (Yang et al., 2024b) and SE-Agent (Guo et al., 2025) deploy professional tool functions and reuse interaction trajectories to construct evolving knowledge bases. Despite these advances, most existing approaches still rely on the monotonic accumulation of raw context, inevitably suffering from information dilution and attention degradation as the interaction horizon expands (Li, 2023).

Agent Memory and Context Compression. To support long-term reasoning, recent research has increasingly focused on memory management and context compression, integrating retrieval-augmented generation (RAG) (Lewis et al., 2020; Jin et al., 2025) to integrate external knowledge. For memory management, some agents (Park et al., 2023; Zhong et al., 2024) utilize persistent memory stores to simulate long-term interactions, while systems like MemGPT (Packer et al., 2023) and LONGMEM (Wang et al., 2023) enhance LLMs with scalable, hierarchical memory architectures. To directly mitigate sequence length, compression methods (Jiang et al., 2023; Li et al., 2023; Wang et al., 2026) prune redundant information at the token level, whereas attention-based approaches (Fang et al., 2025) optimize efficiency through importance-guided selection. Within agent frameworks, most approaches (Kang et al., 2025) employ a tiny auxiliary LLM to dynamically distill or truncate trajectories; others, like MEM1 (Zhou et al., 2025), coordinate between memory selection and the active reasoning process. ContextBudget (Wu et al., 2026) formulates context management as a budget-constrained sequential decision problem and learns when and how to compress interaction history through reinforcement learning. Recent studies (Su et al., 2026; Ye et al., 2025) have also proposed trajectory folding strategies to truncate growing contexts. However, most of these approaches depend on external auxiliary modules or separate processing stages. It not only introduces significant computational overhead, but fundamentally disconnects the compression mechanism from primary agent's active reasoning state.

3 Methodology

We propose **A**uto-**R**egressive **S**tate **M**achine (ARSM), a lightweight training-free architecture that manages context through a repeated cycle of observation, reflection, state update, and action. After receiving the outcome of a previously executed action, the agent generates a reflection on that outcome, metadata for updating its memory, and a proposal for the next action. The controller uses this metadata to update the structured state and applies the configured retention limits when assembling subsequent context. Section 3.1 describes the output format and HAR records, Section 3.2 defines the memory components and their event-conditioned updates, and Section 3.3 explains how the compression-control parameter adjusts memory capacities.

3.1 Auto-Regressive Chain-Structured Trajectory Abstraction

Auto-Regressive (AR) Self-Compressive Space As shown in Figure 2, we define most models' responses as a structured space $\mathcal{Z}_t$, which falls into three components: conclusions from the previous turn, the direction for the next turn, and the specific action to be executed. Specifically, it is formatted as explicit, structured metadata (e.g., progress reflections, status updates, pitfall extractions) auto-regressively, aligned with its standard reasoning steps. Upon the completion of each reasoning turn, the space $\mathcal{Z}_t$ is structurally parsed as $\mathcal{Z}_t \mapsto \langle \tau_t, \mathbf{u}_t \rangle$, where τ_t represents the extracted causal tuple with precise interaction details, such as reflections and next-turn aims, and $\mathbf{u}_t \in \{0,1\}^{4\times 1}$ represents a discrete one-hot routing vector. By mapping the generated status updates into four distinct operational branches (*Ongoing, Milestone, Failure, Success*), $\mathbf{u}_t$ serves as a routing signal to direct the transition mechanics within our state machine.

Micro Hypothesis–Action–Result (HAR) Chain The traditional agent trajectory $\mathcal{T}$ mainly contains three sequential parts: reasoning, action, and output. To strengthen the causal link in reasoning, we separate the generated message and action of the assistant as a detailed sequence of interactions, apart from reasoning:

$$\mathcal{T}_t = \{(R_i, H_{i+1}, A_{i+1}, O_{i+1})\}_{i=1}^{t-1} \tag{1}$$

where **Hypothesis & Aim** (H_i) is derived from the forward-looking portion of the messages and defines the objective for the next action; **Action** (A_i) represents the exact tool invocation command; **Raw Observation** (O_i) contains the tool's raw execution output, which is mostly redundant; and **Result & Reflection** (R_i) is a concise distillation of O_i that provides insights for the next turn. Specifically, the raw observation O_i frequently contains thousands of tokens of logs, which inevitably obscures the crucial causal links between past reasoning and future actions. To resolve this, our framework compresses the raw trajectory into a highly sparsified HAR chain (see Figure 2):

$$\tau_t = (H_t, A_t, R_t) \tag{2}$$

where such chain abstraction is designed to preserve the causal topology of the debugging process while substantially reducing the agent from the token burden of raw environmental feedback.

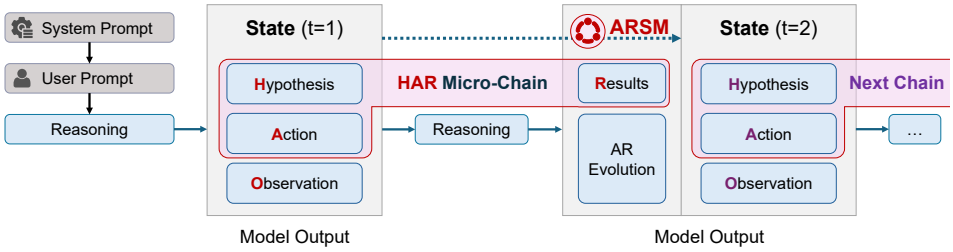

Figure 2: **Workflow of trajectory abstraction.** The agent performs *in-situ* compression: previous raw observations are evaluated and distilled into concise results.

3.2 Dynamic State Machine

The core of our framework is a dynamic state machine that manages the continuous evolution of the agent's internal cognitive state. As illustrated in Figure 1, the system state $\mathcal{S}_t$ evolves through a formal state evolution loop driven by the agent's self-generated metadata:

$$\mathcal{S}_t = \Delta_{\mathcal{Z}_t}(\mathcal{S}_{t-1}) \in \mathbb{S}^{n\times 1} \tag{3}$$

where n is the number of heterogeneous internal state components, and Δ denotes the core transition operator of our dynamic state machine parameterized by the structured metadata space $\mathcal{Z}_t$. Following this update, the newly generated state $\mathcal{S}_t$ with the raw observation is fed back into the agentic model to guide the subsequent reasoning step.

Internal State Definition In our settings, we define the system $\mathcal{S}_t$ as:

$$\mathcal{S}_t = [c_t, p_t, w_t, m_t, a_t]^T \tag{4}$$

where its elements are defined as follows: (i) **Control Layer** (c_t) acts as a global checklist for macroscopic task decomposition and progress tracking; (ii) **Constraint Layer** (p_t, w_t) utilizes pitfall memory (p_t) to prevent redundant failures and loop alerts (w_t) to intervene during circular reasoning; (iii) **Knowledge Layer** (m_t) stores high-level logical milestones that fold completed sub-tasks of macro-chains into verified facts in order to free context; and (iv) **Working Memory Layer** (a_t) maintains the active causal reasoning trajectory of the current micro-chain $\{\tau_i\}$ as compressed HAR tuples, retaining immediate causal context without the overhead of raw logs.

State Transition Matrix Operating alongside the execution loop, the state evolution is governed by the core transition operator Δ. Directed by the previously parsed routing vector $\mathbf{u}_t$, the transition acts as a gate to selectively activate the corresponding operational branch. The unified state transition Δ in Equation 3 is mathematically expanded as a matrix projection:

$$\mathcal{S}_t = \Delta_{\mathcal{Z}_t}(\mathcal{S}_{t-1}) = \Big(\mathbf{T}(\tau_t) \circledast \mathcal{S}_{t-1}\Big)\mathbf{u}_t \tag{5}$$

where $\mathbf{T}(\tau_t) \in \mathbb{O}^{5\times4}$ is the **transition operator matrix** instantiated by the tuple τ_t, and $\circledast$ denotes the **element-wise operator**. It broadcasts the state vector $\mathcal{S}_{t-1}$ into the operator matrix space, such that each atomic operator is applied to its corresponding layer, generating a candidate state matrix. Specifically, $\mathbf{T}(\tau_t)$, constructed of atomic operators (detailed in Appendix A), is defined as:

$$\mathbf{T}(\tau_t) = \begin{bmatrix} \mathbb{I} & \mathbb{I} & \chi(\cdot) & \mathbb{I} & \cdot \oplus \tau_t \\ \delta(\cdot) & \mathbb{I} & \emptyset & \cdot \cup \phi(\tau_t) & \emptyset \\ \mathbb{I} & \cdot \cup \ell(\tau_t) & \mu(\cdot, \tau_t) & \mathbb{I} & \pi(\cdot) \\ \mathbb{I} & \mathbb{I} & \chi(\cdot) & \mathbb{I} & \cdot \oplus \tau_t \end{bmatrix}^T \tag{6}$$

where $\mathbb{I}$ denotes the identity operator that maintains the previous state, and $\emptyset$ denotes a selective truncation operator that discards certain states before milestone τ_t while retaining recent activations.

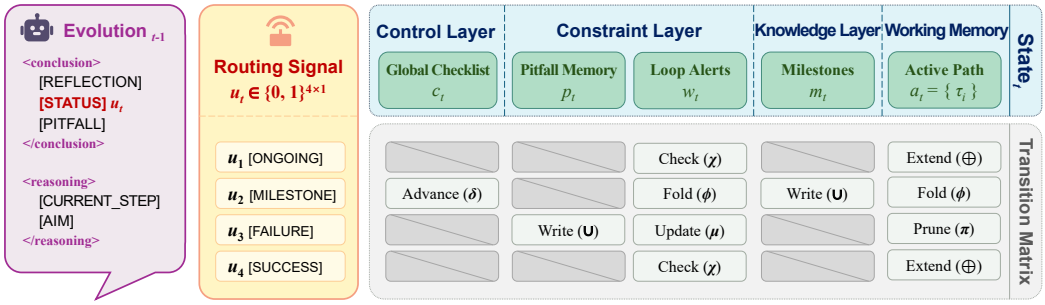

Figure 3: **State transition mechanism.** The model's structured generation space is parsed to yield a discrete routing signal u_t, mapping to the branches. Targeted atomic operators are then dynamically applied to update their respective layers, while unaffected subspaces remain frozen.

With the element-wise broadcasting of $\mathcal{S}_{t-1}$ across the matrix, the gating vector $\mathbf{u}_t$ activates a specific evolutionary branch identified by the judgment block:

- **Ongoing** ($u_1 = 1$): When the model continues to solve a sub-task without yet reaching a definitive conclusion, control and constraint layers remain frozen ($\mathbb{I}$), the active path extends ($\oplus$) with the latest interaction, and loop alerts are checked (χ);
- **Milestone** ($u_2 = 1$): Upon resolving a sub-task, ARSM summarizes acquired knowledge by first folding active paths (ϕ) into the knowledge layer and truncating them ($\emptyset$), and then advancing the checklist (δ) and clearing loop alerts ($\emptyset$);
- **Failure** ($u_3 = 1$): When encountering execution errors or dead ends, to penalize the flawed trajectory and prevent the repetition of identical mistakes, the model extracts failure reasons (ℓ) and updates loop alerts (μ). Invalid active path is then pruned (π) to its last verified node;
- **Success** ($u_4 = 1$): It extends ($\oplus$) with the confirmed result and performs a final loop alert check (χ), marking a valid and successful execution.

Through these mechanisms, ARSM effectively mitigates context bloat and logical contamination, thereby sustaining efficiency and consistency across complex, long-horizon reasoning tasks.

3.3 Sparsified Context Management

ARSM transforms the updated state $\mathcal{S}_t$ into a sparsified context window systematically partitioned into three blocks: (i) **Structured System State** stores compressed context to ensure goal alignment without token bloat; (ii) **Episodic Raw Trajectory** retains recent high-fidelity interactions for execution tracking; and (iii) **Operational Directives** provides explicit instructions to enforce the mandated output schema. To dynamically regulate these block sizes while preserving reasoning continuity, we introduce a continuous compression-control parameter $\rho \in [0, 1]$. The dynamic window size $W(\rho)$ for a given memory component is defined as:

$$W(\rho) = \begin{cases} L_{\max} & \rho = 0 \\ \max\left(V_{\min}, \min\left(L_{\max}, V_{\min} + \left\lfloor (V_r - V_{\min}) \cdot \frac{1-\rho}{\rho} \right\rceil\right)\right) & \rho > 0 \end{cases} \tag{7}$$

where V_r is the target capacity at $\rho = 0.5$, $V_{\min}$ sets the minimum operational fidelity floor for the component, $L_{\max}$ denotes the global system context limit, and $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. More details are listed in Appendix B.

4 EXPERIMENTS

4.1 DATASETS AND BASELINES

We evaluate ARSM on three datasets: WebShop (Yao et al., 2022), a simulated e-commerce website environment with real-world products and crowd-sourced text instructions; Multi-Objective Multi-Hop QA (Yang et al., 2018; Ho et al., 2020; Trivedi et al., 2022; Aksitov et al., 2023) following MEM1 (Zhou et al., 2025); and SWE-Bench Lite (Jimenez et al., 2024), which comprises 300 GitHub repository issues. These settings cover software issue debugging, web navigation, tool interaction, and multi-hop reasoning. We employ Qwen2.5-7B-Instruct, Qwen2.5-14B-Instruct (Team, 2024; Yang et al., 2024a), Qwen3.5-9B, Qwen3.5-27B (Qwen Team, 2026), DeepSeek-V4.1-Flash (DeepSeek-AI, 2026), and GLM-5.3-Flash (GLM-5-Team et al., 2026) as our backbone models. Detailed experimental settings and evaluation metrics are illustrated in Appendix C.1 and C.2.

4.2 COMPARED METHODS

We compare ARSM against several baselines: Search-R1 (Jin et al., 2025), SWE-Agent (Yang et al., 2024b), SE-Agent (Guo et al., 2025), and Agent-FLAN (Chen et al., 2024) are reasoning-driven methods; SWE-Pruner (Wang et al., 2026), MEM1 (Zhou et al., 2025), and ContextBudget (Wu et al., 2026) are memory-aware methods for efficient reasoning.

Table 1: **Comparison of training costs, structural dependencies, and generalizability across methods.** "○" indicates that the generality requires task-specific training.

Method	Training-Free	Self-Contained	Plug-and-Play	Generality
Search-R1	× (RL)	✓	×	× (QA)
MEM1	× (RL)	× (RAG)	×	○
ContextBudget	× (RL)	✓	×	○
Agent-FLAN	× (SFT)	✓	×	○
Mini SWE-Agent	✓	✓	✓	× (Code)
SE-Agent	✓	✓	✓	× (Code)
SWE-Pruner	✓	× (Auxiliary)	×	× (Code)
ARSM (Ours)	✓	✓	✓	✓

As summarized in Table 1, ARSM operates under a lightweight regime, whereas several baselines rely on fine-tuning, reinforcement learning, retrieval augmentation, or auxiliary models, giving them access to additional optimization or inference capacity. Since no prior method operates under exactly the same regime as ARSM, we include them as comparative reference points, making the comparison conservative for ARSM.

4.3 MAIN RESULTS

In this section, we first examine whether ARSM improves the balance between task quality and token usage in each evaluation setting. We report these outcomes jointly, since lower token usage may accompany either improved performance or reduced accuracy. Section 4.4 then examines the behavior of the memory controller, compares simplified configurations, and studies sensitivity to the compression-control parameter.

WebShop As shown in Table 2, ARSM improves task reward while substantially reducing token consumption, not confined to a single backbone family: it appears on Qwen, DeepSeek, and GLM.

Table 2: Performance on WebShop.

Method	Reward ↑	Peak Token (k) ↓	Total Token (k) ↓
Qwen2.5-7B-Instruct	0.2858	5.41 ± 2.40	63.24
Agent-FLAN-7B (SFT)	0.2342	5.84 ± 1.61 (↑ 7.9%)	63.26 (↑ 0.03%)
MEM1-Webshop (Qwen, RAG+RL)	0.1528	1.55 ± 0.14 (↓ 71%)	21.82 (↓ 65%)
ARSM (Qwen-7B, $\rho = 0.3$)	0.4906	3.61 ± 0.87 (↓ 33%)	21.46 (↓ 66%)
ARSM (Qwen-7B, $\rho = 0.5$)	0.4641	3.01 ± 0.47 (↓ 44%)	23.11 (↓ 63%)
ARSM (Qwen-7B, $\rho = 0.7$)	0.4186	2.51 ± 0.23 (↓ 54%)	20.88 (↓ 67%)
Qwen2.5-14B-Instruct	0.4464	5.46 ± 4.23	45.34
ARSM (Qwen-14B, $\rho = 0.3$)	0.5881	3.84 ± 1.66 (↓ 30%)	24.05 (↓ 47%)
ARSM (Qwen-14B, $\rho = 0.5$)	0.5736	3.23 ± 1.05 (↓ 41%)	21.01 (↓ 54%)
ARSM (Qwen-14B, $\rho = 0.7$)	0.5582	2.70 ± 0.62 (↓ 51%)	20.77 (↓ 54%)
Deepseek-V4.1-Flash	0.6151	18.24	33.90
ARSM (Deepseek, $\rho = 0.5$)	0.6633	11.89 (↓ 35%)	20.04 (↓ 41%)
GLM-5.3-Flash	0.6441	28.53	37.15
ARSM (GLM, $\rho = 0.5$)	0.7025	11.33 (↓ 60%)	24.25 (↓ 35%)

Multi-Objective Multi-Hop QA (16-Objective) As shown in Table 3, ARSM substantially improves both Σ_{EM} and Σ_{F1} (sums over 16 objectives) while maintaining compact context. Across different compression-control parameters, ARSM consistently outperforms other methods.

Table 3: **Performance on Multi-Objective Multi-Hop QA (16-Objective).** N/R denotes results that were not reported in the original work.

Method	Σ_{EM} ↑	Σ_{F1} ↑	Peak Token (k) ↓	Total Token (k) ↓
Qwen2.5-14B-Instruct	0.567	0.703	3.84	N/R
Search-R1 (RL)	0.009	0.011	2.09 (↓ 46%)	N/R
MEM1-QA (14B, truncate)	0.44	0.50	1.19 (↓ 69%)	N/R
ContextBudget[1](30B, w/o RL)	0.17	N/R	8.00 (↑ 108%)	N/R
ARSM (14B, $\rho = 0.3$)	1.85	2.15	0.78 (↓ 80%)	55.06
ARSM (14B, $\rho = 0.5$)	1.87	2.30	0.81 (↓ 79%)	54.09
ARSM (14B, $\rho = 0.7$)	2.01	2.34	0.79 (↓ 79%)	51.43

SWE-Bench Lite As shown in Table 4, ARSM consistently establishes a controllable performance, efficiency trade-off across different model scales. We also analyze the token consumption and interaction turns in Figure 4, indicating reduced prompt payload and accelerated reasoning.

Table 4: Performance on SWE-Bench Lite.

Method	Qwen3.5-9B			Qwen3.5-27B	
	Pass@1 ↑	Pass@5 ↑	Token (M) ↓	Pass@1 ↑	Token (M) ↓
Mini SWE-Agent	26.6%	47.5%	6.57	51.7%	1.74
SE-Agent[2](Evolution)	15.0%	N/A	N/A	N/A	N/A
SWE-Pruner (Auxiliary)	40.7%	58.7%	2.29 (↓ 65%)	45.0%	2.37 (↑ 36%)
ARSM ($\rho = 0.1$)	29.7%	46.67%	2.40 (↓ 63%)	46.5%	0.92 (↓ 47%)
ARSM ($\rho = 0.5$)	26.2%	44.5%	2.05 (↓ 69%)	46.0%	0.84 (↓ 52%)
ARSM ($\rho = 0.9$)	20.3%	53.3%	1.15 (↓ 82%)	38.0%	0.47 (↓ 73%)

[1]ContextBudget reports only the context budget; therefore, we use it as its peak token usage.

[2]We were unable to complete the evaluation for SE-Agent; its complex system prompts and tool-calling requirements overwhelmed the models, consistently resulting in format errors or empty patches.

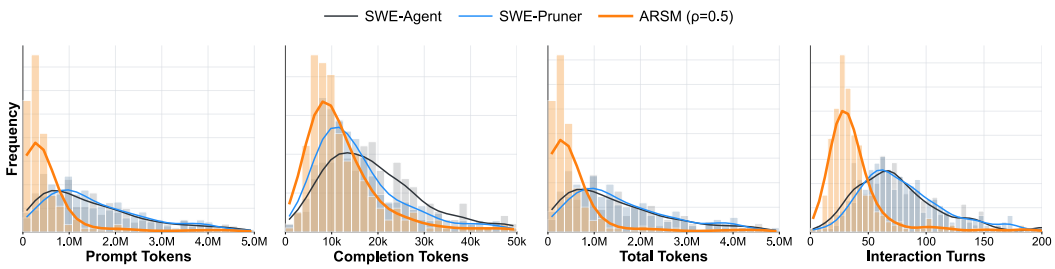

Figure 4: **Efficiency analysis on SWE-Bench Lite using Qwen3.5-27B.** It illustrates the distribution of tokens and interaction turns. Our ARSM ($\rho = 0.5$) consistently shifts the distribution towards the left, demonstrating higher efficiency compared to baselines.

4.4 Ablation Studies

Compression Control To investigate the sensitivity of our core mechanism, we conduct an ablation study on the compression-control parameter ρ. Figure 5 shows that ρ governs the trade-off between the retention of contextual details and the aggressiveness of token pruning.

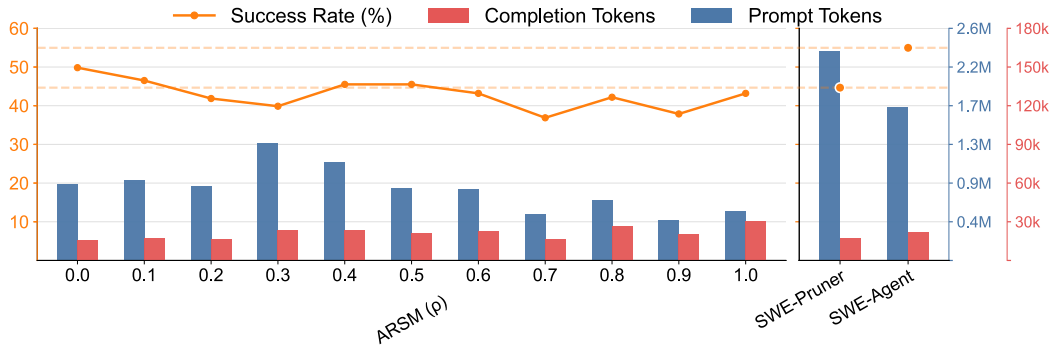

Figure 5: **Performance and efficiency comparison across different ARSM compression-control parameter (ρ) and baselines using Qwen3.5-27B.** ARSM drastically reduces prompt token usage compared to SWE-Agent and SWE-Pruner across all evaluated compression rates.

Routing Behavior Analysis. As shown in Table 5, the routing behavior of ARSM ($\rho = 0.5$) indicates that it actively invokes milestone consolidation, failure handling, and completion transitions rather than collapsing into an append-only running memory.

Table 5: **Distribution of routing signals on SWE-Bench Lite, Qwen3.5-9B.**

Routing Signal	Ongoing	Milestone	Failure	Success
Percentage (%)	76.34	8.99	9.17	5.50

Trajectory Summarization and State Machine Evolution We compare ARSM with several simplified variants on Qwen2.5-7B-Instruct, as shown in Table 6. Structured summary compresses aggressively but suffers substantial performance degradation. HAR-only mostly improves task performance, while SM-only mainly reduces memory cost with limited performance change. Combining both yields the strongest performance–efficiency trade-off. More details are provided in Appendix C.4.

Trajectory-Level Evaluation. We evaluate ARSM ($\rho = 0.5$) using an LLM-based evaluator. As Table 7 shows, ARSM preserves factual fidelity and decisive information while maintaining routing and trajectory coherence. Details are provided in Appendix C.4.

Table 6: **Trajectory summarization and state machine (SM) evolution on 200 Webshop instances.** It highlights the complementary roles of HAR abstraction and dynamic state evolution in balancing task performance and token efficiency.

Method	HAR	SM	Reward ↑	Peak Tokens (k) ↓	Total Tokens (k) ↓
Plain Summary	×	×	0.3305	3.63 ± 0.84	44.71
Structured Summary	×	×	0.1321	1.37 ± 0.10	18.44
HAR-only	✓	×	0.4240	3.18 ± 1.86	28.92
SM-only	×	✓	0.3178	2.74 ± 0.23	24.18
ARSM ($\rho = 0.5$)	✓	✓	0.4899	2.97 ± 0.45	22.84

Table 7: **Trajectory-level evaluation of ARSM state updates and routing decisions on 100 WebShop trajectories with $\rho = 0.5$.** Higher scores indicate better performance.

Metric	Score ↑	Interpretation
State Fidelity	76.1%	Factual faithfulness of compressed states
State Sufficiency	81.4%	Retention of information needed later
Routing Correctness	73.2%	Correctness of routing signals
Trajectory Coherence	72.5%	Global consistency across the trajectory

More Experiments. We also conduct the following studies, and those details are provided in Appendix C.3, D, and E.

- **Efficiency and compression-control analysis on Qwen3.5-9B, SWE-Bench Lite.** ARSM reduces token and turn costs, while ρ controls the retention–compression trade-off.
- **Solved-set overlap.** We examine solved instances across different ρ settings, revealing a stable solving core and complementary coverage across compression regimes.
- **State representation analysis.** We illustrate the two complementary memory surfaces of ARSM: recent raw interaction memory and compressed internal state, showing how they respectively preserve local interaction fidelity and long-range reasoning continuity.
- **Compression-control parameter case study.** Using the same instance under different ρ settings, we qualitatively examine how increasing compression shifts task-relevant information from transcript-level raw interaction history toward structured internal state.

5 Conclusion

In this work, we identify monotonic accumulation of raw context as a fundamental bottleneck for LLM-based agents in interleaved reasoning. To overcome the resulting attention sink and computational bloat, we introduce the **A**uto-**R**egressive **S**tate **M**achine (ARSM), a lightweight, self-contained framework for in-situ context compression. By abstracting raw interactions into **H**ypothesis-**A**ction-**R**esult (HAR) micro-chains and orchestrating them through a dynamic state machine, ARSM shifts the agentic paradigm from passive log accumulation to structured state evolution.

Our systematic evaluation across WebShop, Multi-Objective Multi-Hop QA, and SWE-Bench Lite demonstrates that ARSM can substantially reduce context and token costs while improving strong long-horizon reasoning performance in most evaluated settings. Crucially, these efficiency gains are obtained without training or auxiliary compression models, relying instead on the backbone model's intrinsic reasoning and summarization capabilities. Furthermore, by formalizing a compression-control parameter, ARSM provides a tunable mechanism to regulate the context, empowering engineers to explicitly balance operational fidelity against inference cost based on task demands and model capacities.

Overall, our findings suggest that structured state evolution is a promising alternative to continuously expanding interaction histories for long-horizon agents. By autonomously distilling, folding, and pruning working memory, ARSM provides a scalable and cost-effective foundation for deploying next-generation autonomous agents in complex environments and beyond.

AI USE STATEMENT

In this work, we used generative AI tools to assist with writing, including polishing the manuscript, drafting sections of the paper, and improving the presentation of research ideas. We also used generative AI tools to support literature search, including identifying related work and relevant baselines. We did not use generative AI tools to generate synthetic datasets, prove mathematical claims, conduct experiments, or perform other aspects of the research beyond those described above.

All AI-assisted content was carefully reviewed and verified by the authors. We will take responsibility for all the final content of this work.

REPRODUCIBILITY STATEMENT

Implementation details are provided in the Appendix, while the complete source code, YAML configurations, and supporting files required to reproduce the experiments reported in this paper are included in the Supplementary Material.

REFERENCES

Renat Aksitov, Sobhan Miryoosefi, Zonglin Li, Daliang Li, Sheila Babayan, Kavya Kopparapu, Zachary Fisher, Ruiqi Guo, Sushant Prakash, Pranesh Srinivasan, Manzil Zaheer, Felix Yu, and Sanjiv Kumar. Rest meets react: Self-improvement for multi-step reasoning llm agent, 2023. URL https://arxiv.org/abs/2312.10003.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-flan: Designing data and methods of effective agent tuning for large language models. *arXiv preprint arXiv:2403.12881*, 2024.

DeepSeek-AI. Deepseek-v4.1-flash: Pushing the limits of kv cache compression, 2026.

Yixiong Fang, Tianran Sun, Yuling Shi, and Xiaodong Gu. Attentionrag: Attention-guided context pruning in retrieval-augmented generation. *arXiv preprint arXiv:2503.10720*, 2025.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Yifu Guo, Jiaye Lin, Huacan Wang, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, and Mingguang Chen. Se-agent: Self-evolution trajectory optimization in multi-step reasoning with llm-based agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. URL https://www.aclweb.org/anthology/2020.coling-main.580.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 13358–13376, 2023.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. Acon: Optimizing context compression for long-horizon llm agents. *arXiv preprint arXiv:2510.00615*, 2025.

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

Yucheng Li. Unlocking context constraints of llms: Enhancing context efficiency of llms with self-information-based content filtering. *arXiv preprint arXiv:2304.12102*, 2023.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 6342–6353, 2023.

Jinghao Luo, Yuchen Tian, Chuxue Cao, Ziyang Luo, Hongzhan Lin, Kaixin Li, Chuyi Kong, Ruichao Yang, and Jing Ma. From storage to experience: A survey on the evolution of llm agent memory mechanisms. 2026.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

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

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Jin Su, Runnan Fang, Yeqiu Li, Xiaobin Wang, Shihao Cai, Pengjun Xie, Ningyu Zhang, and Fajie Yuan. U-fold: Dynamic intent-aware context folding for user-centric agents. *arXiv preprint arXiv:2601.18285*, 2026.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 2022.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36:74530–74543, 2023.

Yuhang Wang, Yuling Shi, Mo Yang, Rongrui Zhang, Shilin He, Heng Lian, Yuting Chen, Siyu Ye, Kai Cai, and Xiaodong Gu. Swe-pruner: Self-adaptive context pruning for coding agents. *arXiv preprint arXiv:2601.16746*, 2026.

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

Yong Wu, YanZhao Zheng, TianZe Xu, ZhenTao Zhang, YuanQiang Yu, JiHuai Zhu, Chao Ma, BinBin Lin, BaoHua Dong, HangCheng Zhu, RuoHui Huang, and Gang Yu. Contextbudget: Budget-aware context management for long-horizon search agents, 2026. URL https://arxiv.org/abs/2604.01664.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL https://arxiv.org/abs/1809.09600.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *ArXiv*, 2022.

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

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, et al. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*, 2025.

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

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841.

A ATOMIC OPERATOR DEFINITION

To rigorously formalize the internal mechanics of our state machine, we need to establish the semantic definitions of its foundational building blocks. These atomic operators dynamically manipulate different memory subspaces of the system state $\mathcal{S}$ and subsequently construct the global transition matrix:

Advancement Operator [$\delta : \mathcal{C} \to \mathcal{C}$]. It manages the macroscopic task progression during the milestone state. Upon the success of a milestone, δ pops the completed sub-task $c^{(0)}$ from the head of the checklist queue c, formally defined as:

$$\delta(c) = c \setminus \{c^{(0)}\} \tag{8}$$

Folding Operator [$\phi : \mathcal{A} \to \mathcal{M}$]. As a cognitive compression mechanism triggered during a milestone, it projects a high-dimensional reasoning trajectory, i.e., the active path a, into a condensed semantic knowledge representation:

$$\phi(a) = \mathcal{F}_{\text{comp}} \left(\bigoplus_{i=1}^{|a|} \tau_i \right) \tag{9}$$

where $\mathcal{F}_{\text{comp}}$ denotes the auto-regressive abstraction function that semantically synthesizes the sequential HAR tuples. To preserve short-term context and ensure reasoning continuity, the folding operation selectively retains the most recent HAR micro-chains within the active memory rather than compressing the entire trajectory.

Extraction Operator [$\ell : \mathcal{T} \to \mathcal{P}$]. Upon entering a failure state, the framework first initiates a distillation stage to convert raw error data into actionable knowledge. The extraction operator $\ell(\tau_t)$ auto-regressively analyzes the rejected tuple τ_t to deduce the underlying technical cause.

Instead of merely recording the error, it generates a concise instructional synthesis besides the model-generated pitfall memory p. For instance, rather than appending raw traceback logs, it produces a high-level heuristic: "`Avoid repeatedly reading the same directory; pivot to test-case analysis to verify current assumptions.`" This ensures that the essence of the failure is preserved in a format that directly informs future decision-making.

Update Operator [$\mu : \mathbb{W} \times \mathcal{T} \to \mathbb{W}$]. The system will perform loop monitoring to identify and penalize repetitive, flawed behaviors. The update operator μ dynamically increments a loop weight w_t based on the similarity between the current failure and historical trajectory nodes:

$$\mu(w_{t-1}, \tau_t) = w_{t-1} + \gamma \cdot \mathbf{1}_{\text{sim}}(\tau_t, w_{t-1}) \tag{10}$$

where γ denotes the penalty increment and $\mathbf{1}_{\text{sim}}$ is an indicator function. When a repeat failure is detected ($\mathbf{1}_{\text{sim}} = 1$), the framework merges redundant logs into a compact count-based representation (e.g., "`Action [Action] failed` N `times`"). Once w_t exceeds a predefined threshold, the system triggers a structural intervention—such as dynamic prompt switching or hard gating—to forcefully steer the agent away from local minima and toward an alternative reasoning trajectory.

Pruning Operator [$\pi : \mathcal{A} \to \mathcal{A}$]. The system will implement context sanitation to physically reset the reasoning environment when the model is trapped in the failure loop. To prevent invalid or contaminated reasoning chains from bloating the context window and misleading future steps, the pruning operator π executes a defensive rollback on the active path sequence $a = (\tau_1, \ldots, \tau_k)$:

$$\pi(a) = (\tau_1, \ldots, \tau_{k-\eta}) \tag{11}$$

where $\eta \geq 1$ represents the rollback depth to the last verified stable node. Unlike simple truncation, this operation is strategically coupled with the insights from ℓ and μ; the agent resumes from a clean state but carries the newly distilled pitfall memory, ensuring that the subsequent exploration does not replicate the previously pruned logic branches.

Check Operator [$\chi : \mathcal{W} \to \{\texttt{SAFE}, \texttt{WARN}\}$]. It acts as an outer observer for routine monitoring evaluation, verifying the current reasoning trajectory against the loop alerts w to proactively monitor for loop behaviors. It can be formulated as a thresholding function:

$$\chi(w) = \begin{cases} \texttt{WARN}, & \text{if } \exists \omega \in w \text{ s.t. } \omega \geq \theta_{\text{loop}} \\ \texttt{SAFE}, & \text{else} \end{cases} \tag{12}$$

where θ_{loop} represents the maximum tolerable threshold for repetitive behaviors.

B Compression-Control Parameter

Equation 7 uses the scaling factor

$$s(\rho) = \frac{1-\rho}{\rho}, \tag{13}$$

for $\rho > 0$, while $\rho = 0$ is treated separately as the maximum-retention setting, where the capacity is capped only by $L_{\max}$. Thus, smaller ρ values preserve more historical context, whereas larger ρ values impose more aggressive compression.

At $\rho = 0.5$, $s(\rho) = 1$, so the corresponding capacity equals the reference capacity V_r before applying the global upper bound. At $\rho = 1$, $s(\rho) = 0$, and each controlled component reaches its predefined minimum capacity $V_{\min}$.

More concretely, ρ jointly controls the capacities of the *structured system state* and the *episodic raw trajectory*, while operational directives remain fixed. The controlled quantities include:

- **Recent Raw turn Window**: the number of most recent complete turns retained in raw form.
- **Recent Raw Assistant Window**: the number of recent assistant turns whose original action text is preserved.
- **Recent Raw Unclipped Window**: the number of latest turns for which observations are preserved with looser clipping.
- **Active Reasoning Path Window**: the maximum number of reasoning-path records retained in the active state.
- **Post-Milestone Active Path Window**: the tighter cap applied to the active reasoning path after a `MILESTONE` fold.
- **Pitfall Window**: the number of retained failure-derived lessons.
- **Milestone Window**: the number of verified logical milestones retained in compressed state.
- **Online Context Entry Budget**: the maximum number of compressed state entries exposed to the online controller.
- **Output Clipping Budgets**: character-level caps for different observation types, including:
  - general latest output budget,
  - test/traceback budget,
  - diff budget,
  - source/read/view/search budget,
  - other output budget.

Each controlled quantity has its own $(V_r, V_{\min})$ configuration. Therefore, a single compression-control parameter ρ consistently adjusts raw-history retention, structured-state capacity, and observation fidelity, providing a continuous trade-off between context preservation and compression.

C EXPERIMENTS

C.1 EXPERIMENTAL SETUP

Framework Implementation and Configuration. To ensure reproducibility, all experimental parameters are standardized via YAML configuration files. For **WebShop**, we evaluate Qwen2.5-14B-Instruct over 1,000 episodes with temperature $= 0$, a limit of 20 environment steps and 30 model calls per task, and at most 1,024 generated tokens per call. For 7B models, We evaluate with the same limits and 8,192 token context window. The baseline and corresponding ARSM model on Deepseek-V4.1-Flash and GLM-5.3-Flash are accessed directly through third-party APIs. For the **16-Objective Multi-Hop QA**, we assess Qwen2.5-14B-Instruct across 300 instances under zero-temperature setting and 1,024-token budget, permitting up to 20 interaction turns alongside an E5 retriever that fetches the top-3 passages per query. For **SWE-Bench Lite**, to accommodate lengthy reasoning traces, we expand the context horizon for Qwen3.5-9B and Qwen3.5-27B to at most 300 agent steps and 8,192 tokens per call with a sampling temperature of 0.6.

Computational Resources. Our experiments and development were conducted on two primary clusters: one equipped with eight NVIDIA H200 GPUs and another with four NVIDIA RTX A6000 GPUs. Most models were served through vLLM.

C.2 EVALUATION METRICS

To evaluate the performance and efficiency of ARSM and the baseline models, we employ different metrics:

- For Webshop, we measure average final reward and token usage.
- For Multi-Hop QA, we measure exact matching (EM), F1, and token usage.
- For SWE-Bench Lite, we evaluate the models using Pass@1 and Pass@5 and token usage.

These metrics collectively provide a general view of the model's capability, and they are defined as:

Token Usage. To assess the computational cost and resource efficiency, we track the total number of tokens consumed during the entire inference process. It contains both the prompt tokens and the completion tokens. Lower token usage indicates higher inference efficiency and reduced API costs. We also measure peak token usage, the maximum context window size reached at any single point during the model's reasoning or generation process.

WebShop Reward. The WebShop reward measures how well the purchased product satisfies the user's shopping instruction. It considers the product attributes, purchase options, price constraint, and product type:

$$R = r_{\text{type}} \frac{N_{\text{attr}} + N_{\text{option}} + I_{\text{price}}}{N^*_{\text{attr}} + N^*_{\text{option}} + 1}, \tag{14}$$

where N_{attr} and N_{option} are the numbers of matched attributes and options, respectively, and $I_{\text{price}} = 1$ if the product satisfies the price constraint and 0 otherwise. The coefficient r_{type} measures whether the selected product has the correct product type. The final reward ranges from $[0, 1]$, and $R = 1$ indicates that all requirements are satisfied.

Exact Match (EM), F1 Score, and Multi-Objective Score. For each objective in Multi-Objective Multi-Hop QA, Exact Match (EM) measures whether the generated prediction exactly matches the ground-truth answer, while the F1 score measures token-level overlap between the prediction and the reference answer:

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{15}$$

Following the 16-objective Multi-Hop QA setting, we evaluate each instance over 16 objectives. We report the aggregated score as the sum of the scores across all objectives:

$$\text{Score} = \sum_{i=1}^{16} s_i \in [0, 16], \tag{16}$$

where s_i denotes the score for the i-th objective.

Pass@k. It measures the functional correctness of the generated solutions. We generate n samples for each task ($n \geq k$) and count the number of correct samples c. The unbiased estimator for Pass@k is defined as:

$$\text{Pass@}k = \mathbb{E}_{\text{Problems}} \left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right] \tag{17}$$

In our experiments on SWE-Bench Lite, we report the success rate using $k = 1$ and $k = 5$. These metrics respectively reflect the model's efficiency in surfacing internal knowledge and the actual expansion of its underlying capability frontier.

C.3 More Experiments

Efficiency Distribution on Qwen3.5-9B. To provide a comprehensive understanding of our method's generalization ability, we present additional experimental results focusing on the smaller backbone model, Qwen3.5-9B. Figure 6 illustrates the detailed distribution of computational costs across four dimensions: prompt tokens, completion tokens, total tokens, and interaction turns.

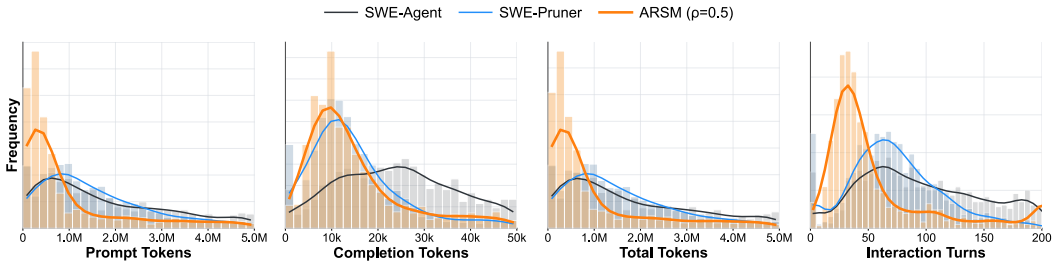

Figure 6: Efficiency analysis on SWE-Bench Lite using Qwen3.5-9B.

Similar to the trends observed with the larger model, the kernel density estimation curves for ARSM ($\rho = 0.5$) exhibit a pronounced leftward shift compared to both SWE-Agent and SWE-Pruner. Notably, the distributions for interaction turns and prompt tokens are highly concentrated in the lower-cost regions. This indicates that even with a weaker reasoning backbone, our state compression mechanism consistently prevents context over-accumulation and guides the agent to reach the correct patch in fewer exploration steps, thereby avoiding the long-tail inefficiency prevalent in standard autonomous agents.

However, a residual long tail in interaction turns remains. We attribute this primarily to the inherent instruction-following limitations of smaller models; specifically, the 9B model occasionally struggles to reliably generate the structured summarizations (e.g., reflections, pitfalls, and aims) required by our prompts, necessitating additional recovery turns.

Impact of Compression-Control Parameter on Qwen3.5-9B. As shown in Figure 7, baseline SWE-Agent consumes about 7.0M tokens, which severely limits its practical deployment. In stark contrast, ARSM introduces a massive reduction in prompt token usage across all evaluated compression rates, keeping the payload strictly under 3.0M tokens. As ρ increases (indicating more aggressive compression), the prompt token cost monotonically decreases.

Crucially, this substantial cost reduction does not lead to a catastrophic collapse in performance. While extremely aggressive compression (e.g., $\rho = 0.9$) inevitably discards some necessary context and causes a drop in the success rate, intermediate values successfully maintain a competitive success rate, showing that ARSM provides a highly effective and tunable trade-off space.

Effect of ρ on Solved-Set Overlap. We analyze the exact overlap of instances solved by ARSM under different ρ settings for both Qwen3.5-9B and Qwen3.5-27B, as shown in Figure 8&9.

Across model scales, ARSM exhibits a clear stable core: 25 instances are solved by all evaluated ρ settings on Qwen3.5-9B, and 48 instances are solved by all 11 ρ settings on Qwen3.5-27B. The

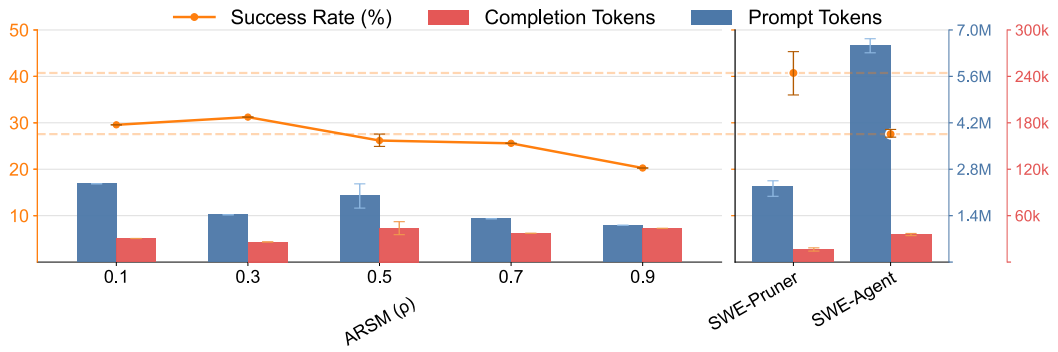

Figure 7: Performance and efficiency comparison across different compression-control parameters (ρ) and baselines using Qwen3.5-9B.

largest intersections are dominated by fully stable or near-stable solutions, where only a small number of ρ values fail to solve the instance. At the same time, both models show a non-trivial long tail of rare intersection patterns, indicating that varying ρ also expands coverage by enabling ARSM to solve additional instances beyond the stable core.

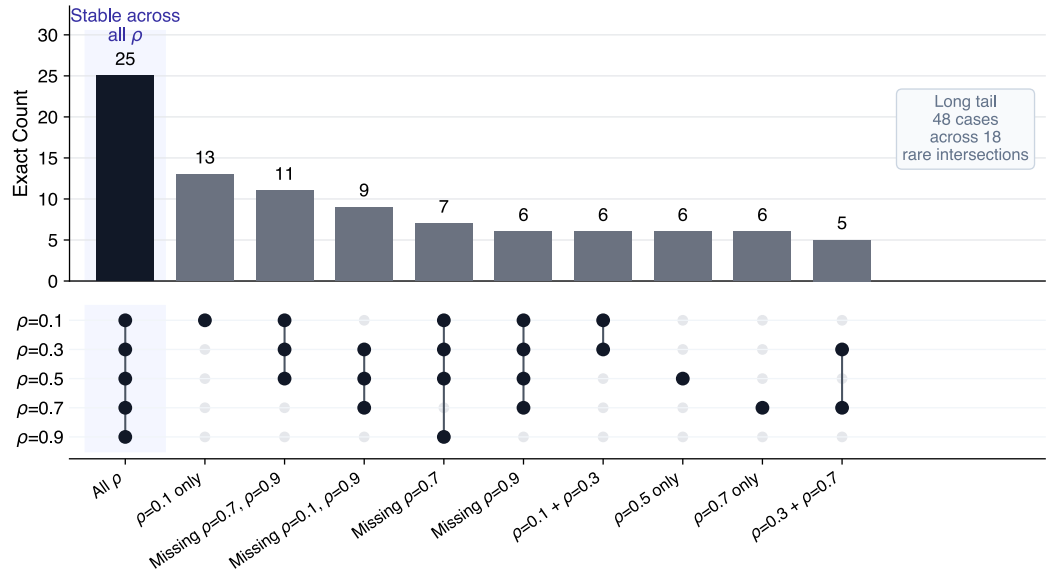

Figure 8: Intersections of resolved SWE-Bench Lite instances on Qwen3.5-9B model.

Overall, these results show that ARSM has a strong shared solving capability across ρ settings, while the variation in ρ further broadens its coverage through complementary exploration.

Table 8: Solved-set overlap of ARSM across different ρ settings.

Model	Union Solved	Solved by All ρ	Long-Tail Coverage
Qwen3.5-9B	142	25	48 cases / 18 patterns
Qwen3.5-27B	190	48	94 cases / 90 patterns

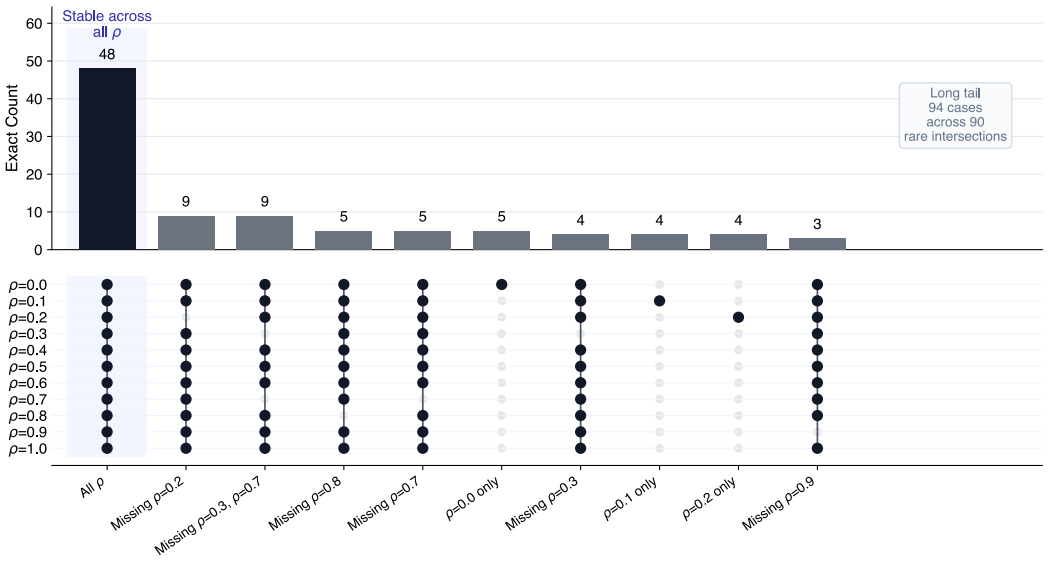

Figure 9: Intersections of resolved SWE-Bench Lite instances on Qwen3.5-27B model.

C.4 Experiment Protocol

Evaluation Protocol of Trajectory Summarization and State Machine Evolution. We evaluate five variants to examine the effects of trajectory summarization and state machine evolution:

- **Plain Summary** repeatedly rewrites the interaction history as a single free-form summary.
- **Structured Summary** rewrites a single summary with fixed fields for the goal, progress, important observations, failures, and next action.
- **HAR-only** accumulates hypothesis–action–result records in chronological order. When the context budget is reached, it removes the oldest complete records.
- **SM-only** retains the dynamic state machine, including its checklist, candidate ledger, pitfalls, milestones, active path, routing signals, and state transitions. It uses generic structured turn summaries instead of HAR records.
- **Full ARSM** combines HAR records with the dynamic state machine.

Trajectory-Level Evaluation Protocol. To evaluate the quality of ARSM's compressed state and routing decisions, we conduct a trajectory-level analysis on 100 WebShop trajectories generated with $\rho = 0.5$. We use DeepSeek-V4.1-Flash (DeepSeek-AI, 2026) as the LLM-based evaluator. For each trajectory, the evaluator assesses every interaction turn independently under four criteria: state fidelity, state sufficiency, routing correctness, and trajectory coherence,

$$s_{j,t}^{(m)} \in \{0, 1\}, \tag{18}$$

where j indexes the trajectory, t indexes the interaction turn, and m denotes the evaluation metric. A score of 1 indicates that the turn satisfies the corresponding criterion, while 0 indicates otherwise. We first compute the average score within each trajectory:

$$\bar{s}_j^{(m)} = \frac{1}{T_j} \sum_{t=1}^{T_j} s_{j,t}^{(m)}, \tag{19}$$

where T_j is the number of turns in trajectory j. We then report the final metric by averaging the trajectory-level scores over all 100 trajectories:

$$S^{(m)} = \frac{1}{100} \sum_{j=1}^{100} \bar{s}_j^{(m)}. \tag{20}$$

The resulting value is reported as a percentage in Table 7. This two-stage aggregation gives equal weight to each trajectory, rather than allowing longer trajectories with more turns to dominate the final score.

D ILLUSTRATION OF STATE REPRESENTATION

We illustrate what ARSM actually stores in its two main memory surfaces. Using the instance `django__django-16408` as a running example, we show how local reasoning traces are represented in (i) recent raw interaction memory, and (ii) compressed internal state.

D.1 RECENT RAW INTERACTION MEMORY

Recent raw memory preserves a bounded portion of the original interaction trajectory in a near-transcript form. Each preserved segment typically contains four elements: Hypothesis, Action, Observation and Result.

In our implementation, this memory surface is formatted as a lightweight HAR chain record. A representative example from `django__django-16408` ($\rho = 0.5$, Turn 20) is shown below:

ARSM Trajectory Example

```
<conclusion>
[REFLECTION] Where: Reading query_utils.py Result: The output was
    truncated, showing only line numbers and summaries. The test
    failure shows that when using FilteredRelation with
    select_related, the wrong related object is returned (PoolStyle
    instead of Tournament). Conclusion: Need to see the full
    FilteredRelation class implementation to understand the bug.
[STATUS] ONGOING
[PITFALL] N/A
</conclusion>
<reasoning type=\"CONTINUE\">
[CURRENT_STEP] Understand current logic
[AIM] Read the full FilteredRelation class implementation from
    query_utils.py
</reasoning>
<mswea_bash_command>
\ngrep -n \"class FilteredRelation\" /testbed/django/db/models/
    query_utils.py
</mswea_bash_command>
```

This representation is intentionally close to the original interaction trace. It is useful for preserving local procedural detail, especially when the model may still need to reuse exact command outcomes, recently opened file fragments, or immediate debugging evidence.

D.2 COMPRESSED INTERNAL STATE

In contrast, the compressed internal state is not a transcript replay. Instead, it reorganizes the trajectory into a structured summary that retains reasoning continuity while discarding low-value interaction redundancy.

A representative internal-state excerpt for `django__django-16408` ($\rho = 0.5$, Turn 50) is shown below:

ARSM Internal State Example

```
[Internal State Update]
# Global Checklist
```

```
[x] 1. Exploring the codebase structure to understand the issue
[ ] 2. Understand current logic
[ ] 3. Identify root cause
[ ] 4. Patch the source
[ ] 5. Verify / submit

# Pitfall
- Need to use Django's test infrastructure instead of direct Python
    imports.
- Django test runner location needs to be discovered.
- Django needs to be installed in development mode from the testbed
    directory before running tests.
...

# Milestones
- No verified milestones yet.

# Loop Alerts
- Inspect loop warning: the current step is repeatedly reading the
   same file/path set 3 times without new evidence. Stop inspecting
    this target; if a patch exists inspect `git diff -- .` and
   submit with `echo COMPLETE_TASK_AND_SUBMIT_FINAL_OUTPUT`,
   otherwise edit the target or mark FAILURE and pivot.
- Execution guard active: repeated looping has blocked this command
    path. You must change the step, file target, or action type.
...

# Active Reasoning Path
- Turn 1: [Hypothesis] Find the relevant files for FilteredRelation
    and the test case [Action] find /testbed -type f -name \"*.py\"
    | xargs grep -l \"FilteredRelation\" | head -20 </mscontent> [
   Reflection] /testbed/django/db/models/query.py /testbed/django/
   db/models/query_utils.py /testbed/django/db/models/sql/query.py
   /testbed/django/db/models/__init__.py /testbed/tests/
   filtered_relation/tests.py /testbed/tests/
   select_related_onetoone/tests.py /testbed/tests/queryset_pickle/
   tests.py /testbed/tests/known_related_objects/tests.py bash: -c:
    line 2: syntax error\u2026. Found relevant files containing
   FilteredRelation. Need to examine the test file to understand
   the issue context.
- Turn 2: [Hypothesis] Read the known_related_objects/tests.py file
    to understand the test and models [Action] cat /testbed/tests/
   known_related_objects/tests.py [Reflection] from django.db.
   models import FilteredRelation from django.test import TestCase
   from .models import Organiser, Pool, PoolStyle, Tournament class
    ExistingRelatedInstancesTests(TestCase): @classmethod def
   setUpTestData(cls): cls.t1 = Tournament.objects.create(name=\"
   Tourney 1\") cls.t2 = Tournament.objects.create(name=\"Tourney
   2\") cls.o1 = Organiser.objects.crea\u2026. Found the test file.
    Need to see the full content including the test case that's
   failing and the models.
...
```

Compared with recent raw memory, this format is more abstract. It no longer stores each interaction as a full transcript fragment. Instead, it preserves the causal reasoning trace in a compressed form that is easier to carry across long horizons.

D.3 Interpretation

The distinction between these two memory surfaces is central to ARSM.

Recent raw memory preserves *interaction fidelity*: it keeps the model close to the original tool-use trace and therefore supports short-horizon procedural reuse. Compressed internal state preserves *reasoning continuity*: it extracts stable conclusions, reusable pitfall lessons, and active hypotheses from that trace, making them available even when the original interaction history has been pruned.

This distinction between structured system state and episodic raw trajectory becomes especially important when the compression-control parameter ρ changes. As ρ increases, ARSM does not merely reduce the amount of remembered history; it shifts a larger fraction of task-relevant continuity from transcript-like raw replay into structured internal state. This effect is examined next in our trajectory comparison case study.

E Case Study: Trajectory Comparison Under Different Compression-Control Parameter

To provide a concrete illustration of how ARSM changes the organization of reasoning history under different compression settings, we present an instance-level case study on `django__django-16408`. We compare the same task under three compression-control parameters: $\rho = 0.2$, $\rho = 0.5$, and $\rho = 0.8$.

E.1 Trajectory Statistics

Table 9 summarizes the overall trajectory lengths. The same task produces substantially different rollout lengths under different compression regimes: the moderate setting $\rho = 0.5$ leads to the longest trajectory, while $\rho = 0.2$ and $\rho = 0.8$ terminate earlier.

Table 9: Trajectory length comparison under different compression-control parameters.

Instance	$\rho = 0.2$	$\rho = 0.5$	$\rho = 0.8$
`django__django-16408`	151	300	207

This difference indicates that ρ is not merely controlling how much history is retained. Instead, it changes the agent's effective reasoning process by shifting the balance between raw interaction replay and structured internal-state accumulation.

E.2 Turn-Level Memory Comparison

To make this shift explicit, we compare several aligned turns from the same instance. For each turn, we focus on two observable quantities: (i) the amount of retained recent raw interaction context, and (ii) the amount of information folded into the internal state update.

Qualitatively, the three settings exhibit distinct patterns:

- At $\rho = 0.2$, the agent keeps a larger amount of recent raw interaction history available, and the internal state grows more conservatively.
- At $\rho = 0.5$, the controller more aggressively folds history into structured state, yielding a much larger internal summary.
- At $\rho = 0.8$, the raw interaction surface is more selective still, and the agent relies more heavily on compact but persistent summarized state.

This behavior is consistent with the intended role of ARSM: as ρ increases, context preservation shifts from transcript-level replay toward state-level continuity.

E.3 Representative Snippets

We next compare representative excerpts from the trajectories. The purpose is not to compare the exact semantic correctness of each intermediate reasoning step, but rather to show how the same underlying task history is represented differently across compression regimes.

Structured System State. The strongest contrast appears in the internal state itself. At $\rho = 0.2$, the state accumulates lessons, but the representation remains relatively restrained. At $\rho = 0.5$, the state becomes much larger and encodes a long sequence of refined hypotheses about how join reuse, nested relation resolution, and FilteredRelation paths should be handled. At $\rho = 0.8$, the state is still substantial, but comparatively more compressed than the $\rho = 0.5$ case.

Episodic Raw Trajectory. In lower-compression settings, recent raw memory retains more direct traces of exploratory interaction. At higher compression levels, the retained raw content becomes more selective and increasingly focused on the active subproblem.

Table 10: **Qualitative comparison of recent raw memory.** As ρ increases, the raw interaction surface becomes more selective and more localized to the current subproblem.

$\rho = 0.2$	$\rho = 0.5$	$\rho = 0.8$
`Recent raw interaction remains broad and preserves more direct traces of earlier file-reading failures, exploratory inspection, and repeated attempts to understand the relation structure around FilteredRelation.`	`Recent raw interaction is still available, but it is more selective; repeated low-value inspection is progressively displaced by folded state, and the retained raw messages are increasingly concentrated on specific subproblems such as join reuse and multi-level relation resolution.`	`Recent raw interaction is narrower still; the visible raw context is more sharply localized to the current debugging hypothesis, while earlier exploration is no longer exposed directly and instead survives primarily through structured memory.`

E.4 Interpretation

This case study illustrates the intended operational effect of ARSM. Increasing ρ does not simply delete history. Instead, it changes *where* the task-relevant information lives.

Under smaller ρ, the model can continue to rely on a larger volume of recent raw interaction history. Under larger ρ, the controller increasingly shifts continuity into explicit state variables, such as lessons, milestones, and active reasoning-path summaries. In other words, ARSM converts history from a transcript-centric representation into a state-centric representation.

The `django__django-16408` example is useful precisely because it shows this transition in a nontrivial long-horizon setting. The three trajectories do not merely differ in length; they differ in memory substrate. This provides direct qualitative evidence that the compression-control parameter controls not only context quantity, but also the form in which reasoning continuity is preserved.

Table 11: **Representative internal-state excerpts.** As compression increases, more of the long-range reasoning trace is preserved through structured pitfalls instead of direct transcript replay.

$\rho = 0.2$	$\rho = 0.5$	$\rho = 0.8$

`# Pitfall`
`- The file-reading commands are producing malformed output; need a more reliable reading method.`
`- Need to use grep or targeted inspection to locate FilteredRelation first.`
`- The problem likely concerns how filtered relation paths are resolved in the ORM.`
	`# Pitfall`
	`- The fix needs to be in setup_joins so that existing joins are reused when the field is already part of a FilteredRelation path.`
	`- The single-level fix is insufficient; nested paths such as tournament_pool.tournament require the second-level relation to use the correct join.`
	`- The necessary information may need to be propagated through the relation metadata structure rather than patched only at the surface.`
		`# Pitfall`
		`- The current fixes address part of the join path but still cache the intermediate object on the wrong model.`
		`- The issue appears deeper in how nested filtered relations are represented in the relation chain.`
		`- Submission may need to avoid unrelated test edits and focus on source-level correction only.`