FLOWSTATE: EXECUTION STATE AS MEMORY FOR LONG-HORIZON LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon tasks require LLM agents to continually draw on information from earlier interactions. However, retaining the full history increases context costs, while compressing it risks losing details needed later, and the relevance of historical information often becomes apparent as the task progresses. To address these challenges, we propose **FlowState**, which treats execution state as memory that can be retained and revisited across requests, unifying current decision-making with the reuse of historical information. FlowState preserves semantically typed state nodes, their relations, and references to raw tool observations, separating persistent retention from on-demand access. Within a single execution loop, Incremental State Update (**ISU**) maintains the current state based on new inputs and feedback, while Progressive State Access (**PSA**) progressively reveals historical states and supporting evidence as needed during reasoning. Together, these mechanisms enable agents to reassess prior decisions in light of new information and guide subsequent actions. Compared with a full-context baseline using the same DeepSeek-V4-Flash model, FlowState improves the average success rate on MemoryArena and the average pass rate on τ^3-Bench by **4.55** and **13.95** percentage points, respectively, while reducing total token consumption by **43.2%** and **40.6%**. These results demonstrate the performance and efficiency advantages of FlowState on long-horizon tasks.

1 INTRODUCTION

As large language models improve in reasoning, tool use, and environmental interaction, LLM-based agents can complete complex tasks through multiple steps of action and observation (Yao et al., 2022). Long-horizon success, however, requires agents to track execution progress and use accumulated information to guide subsequent actions (He et al., 2026). Travel arrangements must respect existing constraints, while mathematical derivations may depend on earlier intermediate results. Such dependencies persist across subtasks and requests, making it essential for information acquired during execution to remain available for future decisions.

Maintaining this information poses a trade-off. Full interaction histories increase context costs and can obscure relevant details (Singh et al., 2026), while compressing these histories may discard information needed for subsequent actions (Kang et al., 2025). State-maintenance methods organize facts, constraints, and progress into working representations (Rozanov & Rei, 2025; Uddin et al., 2026; Guo et al., 2026). ZipAct, for example, replaces the growing history with a continuously updated compact state (Pan et al., 2026). Yet repeated compression may discard details needed later (Ye et al., 2025), whereas retaining historical information in the working state can clutter the context. Agents therefore need to separate persistent information retention from current use.

Long-term memory methods preserve information through external storage and retrieval (Packer et al., 2023; Chhikara et al., 2025; Xu et al., 2026; Rasmussen et al., 2025), but when and what to retrieve often emerge only during reasoning (Ji et al., 2026). New feedback or changing constraints can make earlier information relevant again, and a retrieved record may require further examination of related states or original evidence. In Figure 1, a user books hotel A and later learns that hotel B's price has dropped. Whether to switch depends on both the new price and the earlier booking's cancellation terms. Memory access must therefore unfold alongside decision-making, enabling agents to retrieve and combine historical information as needs emerge.

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

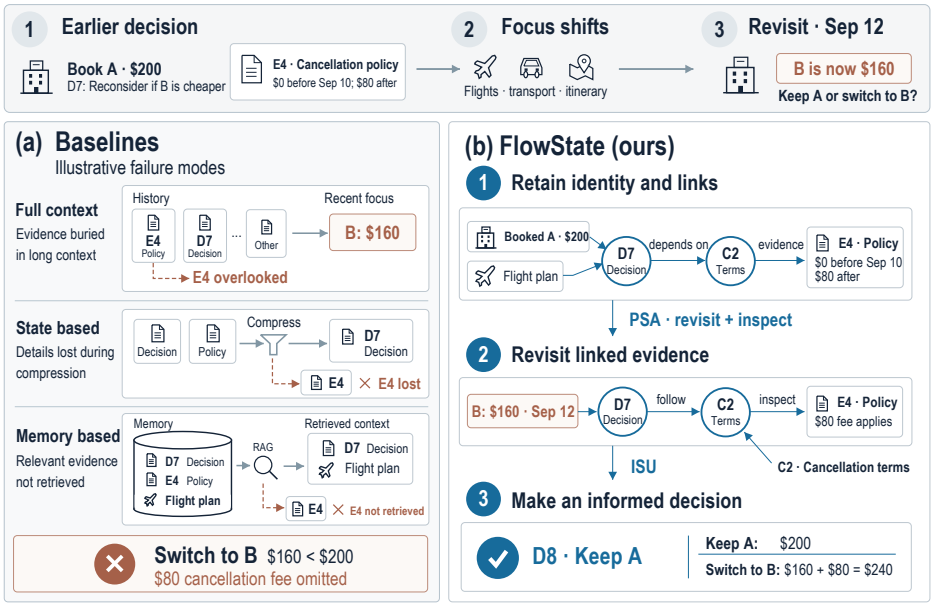


Figure 1: **Revisiting earlier decisions through linked evidence.** A user books hotel A for $200, planning to reconsider if B becomes cheaper, then handles other travel arrangements. On September 12, B drops to $160, but A's $80 cancellation fee makes switching costlier ($240 vs. $200). **(a)** Illustrative baseline failures: the cancellation policy may be overlooked in full context, lost during compression, or stored but not retrieved. **(b)** FlowState preserves D7 and its links to cancellation terms C2 and policy evidence E4 beyond the current view. PSA revisits D7 and linked evidence; ISU combines the new price with this evidence to form an updated decision to keep A (D8).

Recent work uses current state to select historical experiences (Wang et al., 2026) and intermediate evidence to adjust memory access paths (Ji et al., 2026). This motivates our central question: **Can execution state itself serve as a unified representation connecting current decisions with long-term memory?** Retaining execution states across requests and restoring them with related information and original evidence would let agents build on prior work without repeatedly processing the full history. Execution state could thus serve as both a working representation and a reusable memory unit.

We propose FlowState, an agent memory framework centered on persistent execution state. FlowState represents user requirements, environmental information, agent judgements, and execution artifacts as semantically typed nodes, preserving their relations and references to raw tool observations. Full contents persist in a Persistent State Repository, while identifiers and concise summaries form a Historical State Index included in the model context. The agent selectively expands historical states through the same representation used for current states, separating persistent retention from context visibility to keep the working context compact.

FlowState integrates two complementary mechanisms into a single execution loop. Incremental State Update (ISU) creates and revises current states from new inputs and execution feedback without reconstructing the entire state. Progressive State Access (PSA) retrieves historical content through the index or relations in already disclosed states, tracing back to raw observations as needed. Retrieved information informs subsequent actions and state updates. At the end of each request, the resulting states and updated index are retained for future reuse.

We evaluate FlowState on MemoryArena (He et al., 2026) and τ^3-Bench (Barres et al., 2025), covering shopping, travel planning, formal reasoning, and service interactions governed by business rules. Compared with the full-context baseline using the same DeepSeek-V4-Flash model, FlowState improves the average success rate on MemoryArena and the average pass rate on τ^3-Bench by 4.55 and 13.95 percentage points, respectively, while reducing total token consumption by 43.2%

and 40.6%. Further analyses show that ISU and PSA play complementary roles, with strong performance on later subtasks and sustained performance and efficiency gains in evaluated settings with strong baseline models.

Our main contributions are as follows:

- **A unified memory representation centered on execution state.** We extend execution states into memory units that can be reused across requests, connecting decisions with historical information through semantic types, node relations, and evidence references.
- **Incremental maintenance and progressive access within a single execution loop.** ISU and PSA maintain states for the current request and access historical information on demand, allowing memory to accumulate during execution and inform decisions.
- **Improved success rates and token efficiency across diverse long-horizon tasks.** FlowState improves task success rates while reducing token consumption on MemoryArena and τ^3-Bench. We further examine its effectiveness through ablations, analyses across subtask depths, and experiments across base models.

2 FLOWSTATE

We consider a sequential interaction task consisting of user requests $q_1, \ldots, q_K$. For request q_k, the agent interacts with the environment over T_k ReAct decision steps, each corresponding to one model invocation. A fixed policy π_θ makes decisions based on the information available at each step; its parameters remain unchanged throughout the interaction. Constraints, observations, action outcomes, and assessments from earlier requests may become relevant again, while the specific historical information needed often emerges only as the current request unfolds. FlowState therefore persists execution state across requests and provides access to it on demand, allowing historical states to inform subsequent decisions.

2.1 PERSISTENT EXECUTION STATE

FlowState's persistent execution state comprises a runtime-maintained *Persistent State Repository*, a *Historical State Index* for the current request, and an *Active State* that evolves during execution.

Persistent State Repository ($\mathcal{M}_k$). We define $\mathcal{M}_k$ as the persistent state repository at the start of request q_k. It stores prior user requests, state nodes, and relations among nodes. Each node records its source request, semantic identifier, category, and full content. A raw tool result is also retained as an accessible state node if it is referenced by a state newly created by the model. The runtime maintains $\mathcal{M}_k$, but does not place it directly in the model context. At the end of q_k, newly created states and relations retained for subsequent requests are incorporated into $\mathcal{M}_{k+1}$.

Historical State Index (G_k). We define $G_k = \mathrm{Index}(\mathcal{M}_k)$ as a lightweight index of historical states constructed at the start of request q_k.

The index covers all preceding requests $q_1, \ldots, q_{k-1}$. For each request, a state entry contains only its semantic identifier and a concise summary maintained for that state. The lightweight index G_k is included in the model context for request q_k.

Active State ($S_{k,t}$). We define $S_{k,t}$ as the Active State available for decision-making at step t of request q_k. It contains states created during the current request, together with disclosed historical states and their relations. The full contents of undisclosed historical states are excluded from $S_{k,t}$.

Nodes in the Active State follow a taxonomy of five state types. FlowState classifies information by source and role: user-side *preferences* record the user's preferences, requirements, and constraints; environment-side *knowledge* records tool observations, while *attributes* capture entity-level information distilled from those observations; and agent-side *judgements* record the agent's assessments of task objectives, plans, decisions, and risks, while *artifacts* record reusable outputs produced by the agent. Details on the semantic scope of each state type are provided in Appendix B.2.

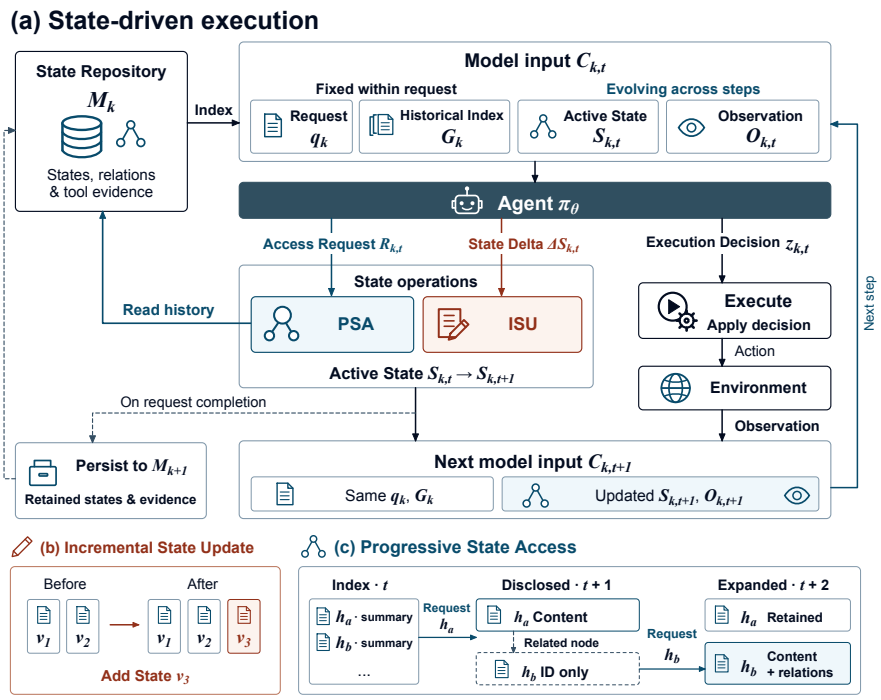


Figure 2: **FlowState execution loop and state operations.** (a) Each model invocation proposes a state delta, a historical-state access request, and an execution decision. ISU and PSA update the Active State; retained states and evidence are persisted at request completion. (b) ISU is illustrated by adding v_3 to the current-request state. (c) PSA discloses h_a, whose revealed relation guides a later access to h_b.

2.2 State Operations

FlowState maintains $S_{k,t}$ through two complementary operations: *Incremental State Update* (ISU) commits local state changes in response to new information, while *Progressive State Access* (PSA) progressively accesses historical states as information needs arise during execution. The model proposes both types of requests from the same context. After validation against $S_{k,t}$ under execution rules P (Appendix B.4), their effects are combined to produce $S_{k,t+1}$. Concurrent work, MAGE (Chen et al., 2026), organizes execution history as a hierarchical state tree with active-path context and branching revision. FlowState instead uses semantically typed state nodes that are individually addressable across requests, separating persistent retention from visibility in the Active State. Through PSA and ISU, historical states and supporting evidence are progressively disclosed to inform current-request state updates.

2.2.1 Incremental State Update (ISU)

At step t, the model uses the current context $C_{k,t}$ to produce a candidate state delta $\Delta S_{k,t}$, comprising three operations on state nodes: Add, Update, and Remove. Add creates a state for the current request; Update and Remove apply only to states created during the current request. The state delta is validated before it is committed:

$$\widehat{\Delta S}_{k,t} = \mathrm{Validate}_{\mathrm{upd}}(\Delta S_{k,t} \mid P, S_{k,t}), \qquad S^{+}_{k,t} = \mathrm{Commit}(S_{k,t}, \widehat{\Delta S}_{k,t}). \tag{1}$$

The validated state delta $\widehat{\Delta S}_{k,t}$ is committed to $S_{k,t}$, yielding the intermediate Active State $S^{+}_{k,t}$ after ISU. The model need not regenerate the entire state at every step. For example, a price observation returned by the environment may update a *judgement* about lodging selection created during the current request, while an unchanged budget *preference* remains intact.

Algorithm 1 FlowState State-Driven Execution Loop.

Require: Requests $q_1, \ldots, q_K$, fixed policy π_θ, execution rules P, and environment tools

Initialize Persistent State Repository $\mathcal{M}_1$
for $k = 1, \ldots, K$ **do**
  $G_k \leftarrow \mathrm{Index}(\mathcal{M}_k)$
  Initialize $S_{k,1} \leftarrow \varnothing$ and $O_{k,1} \leftarrow \varnothing$; set $t \leftarrow 1$
  while q_k is unfinished **do**
    $C_{k,t} \leftarrow \mathrm{Serialize}(q_k, O_{k,t}, G_k, S_{k,t})$
    Sample $y_{k,t} \sim \pi_\theta(\cdot \mid C_{k,t})$ subject to $y_{k,t} \neq (\varnothing, \varnothing, \varnothing)$
    $(\Delta S_{k,t}, R_{k,t}, z_{k,t}) \leftarrow y_{k,t}$
    $\widehat{\Delta S}_{k,t} \leftarrow \mathrm{Validate}_{\mathrm{upd}}(\Delta S_{k,t} \mid P, S_{k,t})$
    $\widehat{R}_{k,t} \leftarrow \mathrm{Validate}_{\mathrm{acc}}(R_{k,t} \mid P, G_k, S_{k,t})$
    $S^+_{k,t} \leftarrow \mathrm{Commit}(S_{k,t}, \widehat{\Delta S}_{k,t})$
    $S_{k,t+1} \leftarrow S^+_{k,t} \cup \mathrm{Disclose}(\mathcal{M}_k, S_{k,t}, \widehat{R}_{k,t})$
    $O_{k,t+1} \leftarrow \varnothing$
    if $z_{k,t}$ is a final response **then**
      Output $z_{k,t}$ and **break**
    else if $z_{k,t}$ is an environment action **then**
      Execute $z_{k,t}$ and receive tool observation $O_{k,t+1}$
    end if
    $t \leftarrow t + 1$
  end while
  $\mathcal{M}_{k+1} \leftarrow \mathrm{Persist}(\mathcal{M}_k, q_k, S_{k,T_k+1}, \mathbf{O}_k)$
end for

2.2.2 Progressive State Access (PSA)

PSA discloses historical state information on demand as information needs arise during execution. At step t, the model uses state identifiers and their summaries in G_k, together with disclosed relations in $S_{k,t}$, to specify a set $R_{k,t}$ of historical states to access; if no historical states are needed, $R_{k,t} = \varnothing$. After the access request is validated against the execution rules, the corresponding states are disclosed from $\mathcal{M}_k$:

$$\widehat{R}_{k,t} = \mathrm{Validate}_{\mathrm{acc}}(R_{k,t} \mid P, G_k, S_{k,t}), \quad S_{k,t+1} = S^+_{k,t} \cup \mathrm{Disclose}(\mathcal{M}_k, S_{k,t}, \widehat{R}_{k,t}). \tag{2}$$

Disclose returns the contents and relations of validated target states $\widehat{R}_{k,t}$ that are not already in $S_{k,t}$; these are added to $S^+_{k,t}$ to obtain $S_{k,t+1}$. Each relation specifies its type and the identifier of the related node, allowing the model to decide whether to access that node in a subsequent step. Disclosing a relation does not reveal the full contents of the related node.

2.3 State-Driven Execution

Figure 2 situates the ISU and PSA operations described above within the state-driven execution loop.

At step t of request q_k, FlowState serializes the current request, the most recent tool observation, the Historical State Index, and the Active State into a model input. Let $y_{k,t} = (\Delta S_{k,t}, R_{k,t}, z_{k,t})$ denote the model output, whose components are the candidate state delta, the target historical states to access, and the execution decision, respectively:

$$C_{k,t} = \mathrm{Serialize}(q_k, O_{k,t}, G_k, S_{k,t}), \quad y_{k,t} \sim \pi_\theta(\cdot \mid C_{k,t}), \quad \text{s.t. } y_{k,t} \neq (\varnothing, \varnothing, \varnothing). \tag{3}$$

Here $C_{k,t}$ is the model input at this step; $O_{k,t}$ is the tool observation produced by the preceding step, or $\varnothing$ if no tool was called; and $z_{k,t}$ is an environment action, a final response, or $\varnothing$. Each of the three output components may be empty individually, but they cannot all be empty simultaneously.

After the model produces its output, ISU and PSA process their respective requests against $S_{k,t}$ and combine their effects to obtain $S_{k,t+1}$. The execution decision $z_{k,t}$ then determines whether to perform an environment action, terminate the current request, or proceed to the next model invocation. Information disclosed by PSA at step t can inform decisions only from the next model invocation onward. For actions that depend on this information, the model can set $z_{k,t} = \varnothing$ and defer the decision to the next step using $S_{k,t+1}$. An empty state delta results in no ISU update, while an empty access request results in no PSA disclosure. Algorithm 1 gives the complete procedure.

Let $\mathbf{O}_k = (O_{k,1}, \ldots, O_{k,T_k+1})$ denote the sequence of tool observations for request q_k. Upon completion of the request, the Persistent State Repository is updated:

$$\mathcal{M}_{k+1} = \mathrm{Persist}(\mathcal{M}_k, q_k, S_{k,T_k+1}, \mathbf{O}_k), \qquad G_{k+1} = \mathrm{Index}(\mathcal{M}_{k+1}). \tag{4}$$

Persist incorporates request q_k and the newly created states and relations retained at the end of the request into $\mathcal{M}_{k+1}$, and retains any raw tool results referenced by those states as *knowledge* nodes. New states record q_k as their source request; historical states retain their original source requests and cannot be deleted. Index then generates G_{k+1} from $\mathcal{M}_{k+1}$, listing state identifiers and their summaries through request q_k, grouped by request, to guide subsequent PSA access.

3 Experiments

3.1 Experimental Setup

We evaluate FlowState on a subset of MemoryArena environments (Bundled Web Shopping, Group Travel Planning, and Formal Reasoning: Math and Physics) (He et al., 2026) and on τ^3-Bench (Airline and Retail) (Barres et al., 2025). The former tests information reuse across interdependent subtasks; the latter tests policy-constrained tool use with simulated users.

We compare FlowState against Full Context (full interaction history), memory-system methods (ReasoningBank (Ouyang et al., 2026), BM25 (Robertson & Zaragoza, 2009), and Mem0), a state-tracking method (ZipAct), and a hybrid method (ZipAct+BM25). All methods use DeepSeek-V4-Flash (DeepSeek-AI et al., 2026) in the primary comparison; Full Context is also evaluated with GPT-5.6-terra (OpenAI, 2026) and Qwen3.5-397B-A17B (Qwen Team, 2026).

MemoryArena reports Success Rate (SR), Progress Score (PS), and soft Progress Score (sPS) for Travel. τ^3-Bench reports Pass Rate, DB Accuracy, and Action Accuracy. We report total prompt and generation tokens (#Tok). Benchmark, scoring, and implementation details appear in Appendix B.5.

3.2 Main Results

FlowState improves task success and progress on MemoryArena. Table 1 shows FlowState leads on eight of nine task performance metrics. On Bundled Web Shopping, FlowState achieves a PS of 42.89%, outperforming the strongest memory-system baseline by 11.78 percentage points. On Group Travel Planning, it reaches a PS of 12.57%, exceeding the best baseline by 5.61 percentage points. FlowState also achieves the best Math PS and the best SR and PS on Physics.

In contrast, directly combining state tracking with historical retrieval does not yield consistent gains: ZipAct+BM25 outperforms both standalone components only on Math PS. FlowState surpasses ZipAct+BM25 on all nine performance metrics, indicating that, on the interdependent multi-session tasks evaluated in MemoryArena, its design of treating execution states as revisitable memory has a consistent performance advantage over this direct combination.

These gains also come with lower context overhead. Across four environments, FlowState reduces total token consumption by 43.2% compared with Full Context using the same DeepSeek-V4-Flash model, showing it can improve task success and progress while keeping context overhead low.

FlowState's benefits extend beyond MemoryArena. To assess generality across benchmarks, we further evaluate FlowState on τ^3-Bench, which differs from MemoryArena in both task format and evaluation criteria. As shown in Table 2, using the same DeepSeek-V4-Flash model, FlowState improves Pass Rate over Full Context by 20.0 and 7.9 percentage points on Airline and Retail,

Table 1: Results on MemoryArena. Performance metrics are reported in percent; #Tok. denotes total token usage. Bold and underlined performance scores indicate the best and second-best results, respectively. Δ denotes changes relative to DeepSeek-V4-Flash with full context (FC): absolute percentage-point changes for performance and relative percentage changes for token usage (negative values indicate savings).

	Bundled Web Shopping			Group Travel Planning				Formal Reasoning					
								Math			Physics		
Method	**SR ↑**	**PS ↑**	**#Tok. ↓**	**SR ↑**	**PS ↑**	**sPS ↑**	**#Tok. ↓**	**SR ↑**	**PS ↑**	**#Tok. ↓**	**SR ↑**	**PS ↑**	**#Tok. ↓**
Full Context													
DeepSeek-V4-Flash	0.00	21.11	52.65M	0.37	3.09	88.43	231.59M	30.00	50.36	40.60M	55.00	72.50	3.68M
GPT-5.6-terra	0.00	33.89	313.95M	–	–	–	–	**42.50**	54.23	14.59M	50.00	62.60	2.76M
Qwen3.5-397B-A17B	0.00	19.56	134.82M	0.37	6.96	60.99	223.70M	30.00	35.27	19.97M	50.00	55.56	2.02M
Memory System													
ReasoningBank	0.00	31.11	92.72M	0.00	3.48	48.13	49.35M	32.50	45.63	7.90M	55.00	63.97	1.88M
BM25	0.67	22.11	32.98M	0.00	5.24	65.09	69.82M	22.50	49.83	10.06M	45.00	60.63	1.27M
Mem0	0.00	25.78	59.36M	0.00	3.37	40.74	107.22M	25.00	41.96	6.61M	35.00	53.57	1.33M
State Tracking													
ZipAct	0.00	17.78	33.98M	0.00	3.81	25.22	201.86M	30.00	50.66	12.28M	55.00	68.54	3.33M
State Tracking + Memory System													
ZipAct+BM25	0.00	16.11	65.92M	0.00	4.29	19.89	175.40M	30.00	53.09	9.85M	40.00	61.18	2.21M
FlowState (Ours)	**5.33**	**42.89**	44.83M	**0.74**	**12.57**	**91.87**	128.93M	37.50	**55.53**	10.96M	**60.00**	**73.96**	1.78M
Δ *vs. V4-Flash (FC)*	+5.33	+21.78	**-14.86%**	+0.37	+9.48	+3.44	**-44.33%**	+7.50	+5.17	**-73.01%**	+5.00	+1.46	**-51.69%**

Table 2: Results on τ^3-Bench using DeepSeek-V4-Flash for both methods. PR, DB, and Act. denote pass rate, database accuracy, and action accuracy (%), respectively; #Tok. denotes total token usage. Bold marks better performance. Δ is relative to Full Context: percentage points for performance and percentage changes for tokens (negative means savings).

	Airline				Retail			
Method	PR ↑	DB ↑	Act. ↑	#Tok. ↓	PR ↑	DB ↑	Act. ↑	#Tok. ↓
Full Context	58.0	70.5	81.4	21.03M	73.7	80.7	**90.4**	23.55M
FlowState (Ours)	**78.0**	**78.0**	**83.8**	8.16M	**81.6**	**86.0**	85.5	18.32M
Δ *vs. Full Context*	+20.0	+7.5	+2.4	**-61.21%**	+7.9	+5.3	−4.9	**-22.24%**

respectively, and DB Accuracy by 7.5 and 5.3 percentage points, while reducing token consumption by 61.2% and 22.2%. Although Action Accuracy on Retail decreases by 4.9 percentage points, task success improves in both domains. These results show that FlowState's benefits extend beyond MemoryArena, providing further evidence of its generality across different types of interactive tasks.

Joint Performance and Efficiency Gains Across Models. To assess whether FlowState's benefits persist across base models, we evaluate it with DeepSeek-V4-Flash, GPT-5.6-terra, and Qwen3.5-397B-A17B, comparing each with Full Context using the same model. Figure 3 shows higher mean SR and PS for all three models, alongside 28.3%–56.3% lower mean relative token usage. These results suggest that FlowState's performance and token-efficiency gains are not specific to a single base model. Per-task results appear in Appendix Table 5.

4 Analysis

4.1 Component Ablation

We compare FlowState with two ablation variants to assess the roles of ISU and PSA. As shown in Table 3, *w/o PSA* removes the Historical State Index and progressive access, instead retaining the full contents of all ISU-generated historical states in the Active State across subtasks. In contrast, *w/o ISU* retains PSA's access mechanism but populates the Historical State Index with historical state nodes assembled by predefined rules at subtask boundaries.

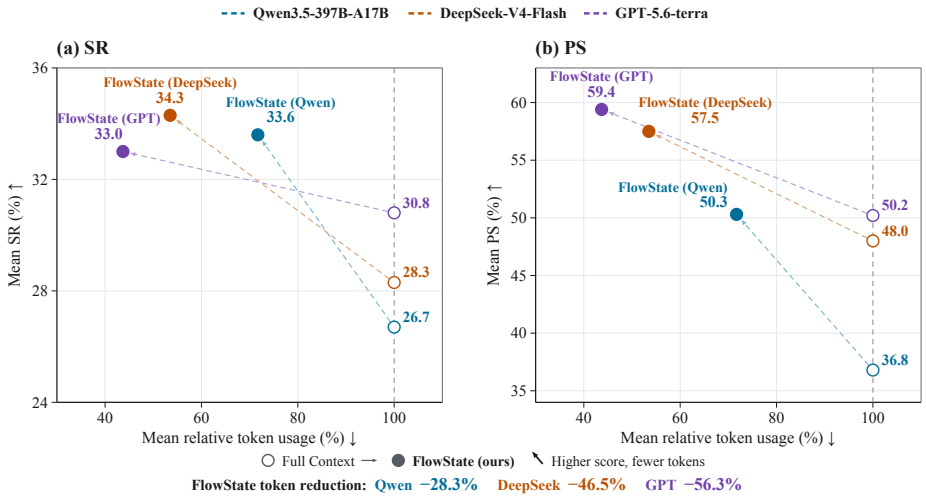


Figure 3: **FlowState consistently improves both performance and token efficiency across base models.** SR and PS are equally weighted averages over Web Shopping, Math, and Physics. Relative token usage is normalized to the corresponding Full Context baseline within each task and then averaged with equal weights across tasks. Hollow and filled circles denote Full Context and FlowState, respectively; dashed arrows connect results for the same base model.

Table 3: Ablation results on Group Travel Planning and Bundled Web Shopping. Performance metrics are in percent; #Tok. denotes total token usage. Bold scores mark the best performance within each environment.

	Travel Planning				Web Shopping		
Variant	SR ↑	PS ↑	sPS ↑	#Tok. ↓	SR ↑	PS ↑	#Tok. ↓
FlowState	**0.74**	**12.57**	**91.87**	128.93M	**5.33**	42.89	44.83M
w/o PSA	0.37	9.36	87.18	160.27M	3.33	**44.67**	115.37M
w/o ISU	0.00	9.15	90.34	93.16M	0.00	32.11	69.01M

ISU improves task performance. Removing ISU reduces SR to zero in both environments, with lower PS as well. With PSA still available, this comparison suggests that states constructed and updated during execution better support task completion than rule-generated historical state nodes at subtask boundaries.

PSA supports task success with lower token usage. Compared with *w/o PSA*, FlowState uses fewer tokens in both environments and improves all performance metrics in Group Travel Planning. Although *w/o PSA* achieves slightly higher PS on Bundled Web Shopping, its SR remains below that of the full model. These results suggest that retaining all historical states does not necessarily improve end-to-end task success. By progressively disclosing relevant states and their supporting evidence as needed, PSA limits irrelevant history in the Active State, balancing task performance with token efficiency.

Overall, on the long-horizon, interdependent multi-session tasks evaluated in MemoryArena, ISU's continuous state construction and PSA's on-demand historical access play complementary roles. The full model shows strong task performance in both environments while using fewer tokens than the variant that exposes all historical states.

4.2 FlowState Sustains Long-Horizon Advantages

By expanding the dependencies and supporting evidence of historical states on demand, PSA enables the agent to continue using earlier information within a limited active context. Figure 4 shows that

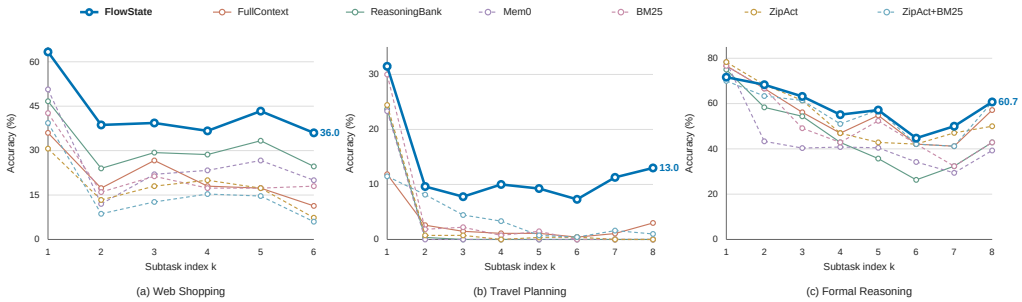


Figure 4: **Performance Across Subtask Depths.** Accuracy on (a) Web Shopping, (b) Travel Planning, and (c) Formal Reasoning. The horizontal axis denotes the subtask index k. The vertical axis reports Acc_k, the percentage of task instances with at least k subtasks whose k-th subtask is completed correctly. Curves represent methods. The blue curve denotes FlowState, with annotations indicating its accuracy at the final reported depth in each panel. Vertical scales differ across panels.

Table 4: Relative performance retention across subtask depths. Metrics use accuracy normalized to each method's initial value. nAUC measures normalized curve area; DR is the fitted decline rate (percentage points per subtask). Best results are bolded and second-best results underlined.

	Web Shopping		Travel Planning		Formal Reasoning	
Method	nAUC ↑	DR ↓	nAUC ↑	DR ↓	nAUC ↑	DR ↓
Full Context	57.22	10.48	18.35	<u>7.30</u>	69.82	4.63
ReasoningBank	<u>64.71</u>	**5.06**	7.37	8.43	58.84	7.08
Mem0	47.11	6.09	7.14	8.33	52.41	5.50
BM25	47.97	8.26	10.14	8.94	64.33	6.52
ZipAct	57.17	9.57	8.48	8.54	68.00	5.57
ZipAct+BM25	37.63	10.61	<u>31.04</u>	12.51	<u>77.87</u>	<u>3.87</u>
FlowState	**65.58**	<u>5.65</u>	**35.17**	**4.66**	**80.66**	**3.68**

FlowState maintains the highest accuracy at every reported subtask depth on Web Shopping and Travel Planning. At later depths on Travel Planning in particular, most baselines approach zero accuracy, while FlowState retains a clear advantage. On Formal Reasoning, FlowState generally leads at later depths and matches ZipAct+BM25 at the final depth.

Table 4 measures relative performance retention using $R_k = 100 \times \mathrm{Acc}_k/\mathrm{Acc}_1$. nAUC is the trapezoidal integral of the normalized trajectory divided by the depth range $K-1$, with higher values indicating better relative performance retention. Decline rate (DR) is the negative least-squares slope of the trajectory with respect to k, with lower values indicating a slower decline. FlowState achieves the highest nAUC in all three domains and the lowest DR on Travel Planning and Formal Reasoning. On Web Shopping, its DR is close to that of ReasoningBank, while its nAUC is higher.

These results show that FlowState combines high absolute accuracy on later subtasks with strong overall relative performance retention, consistent with PSA's design motivation of recovering earlier information on demand through structured access paths to support subsequent decisions.

5 Conclusion

We introduce FlowState, which turns execution states into continually updated, revisitable memory through Incremental State Update (ISU) and Progressive State Access (PSA). Experiments on MemoryArena and τ^3-Bench show that FlowState substantially improves task performance while reducing token consumption, demonstrating the value of unifying state maintenance and historical access for long-horizon decision-making.

AI USE STATEMENT

We used generative AI tools to assist with checking relevant literature, writing research code, creating figures, and polishing the manuscript. The research questions, methodology, experimental design, and interpretation of results were developed by the authors. The authors take full responsibility for the final paper.

ETHICS STATEMENT

Our experiments use benchmark environments and involve no real-user data, human participants or annotators, or real-world transactions. Nevertheless, a system that retains information across interactions could expose sensitive information or carry outdated or incorrect assumptions into later decisions when deployed with real users. Our benchmark evaluation does not establish safety for deployment. Real-world applications would require controls over data access and retention, checks on the provenance and validity of information, and safeguards for consequential actions.

REPRODUCIBILITY STATEMENT

Section 2 describes the state representation and the procedures for updating and accessing state. Section 3.1 specifies the benchmark domains, baselines, and evaluation metrics. Appendix B.4 details the state transition rules, Appendix B.3 presents the runtime prompt templates, and Appendix B.5 reports model settings, benchmark revisions, prompt construction, and baseline implementations. We will release the complete source code at an appropriate time.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Hanxiang Chao, Yihan Bai, Rui Sheng, Tianle Li, and Yushi Sun. Stale: Can llm agents know when their memories are no longer valid? *arXiv preprint arXiv:2605.06527*, 2026.

Yaoqi Chen, Haibin Lai, Yuru Feng, Chuyu Han, Qianxi Zhang, Baotong Lu, Menghao Li, Xinjiang Wang, Zhirui Wang, Shusen Xu, et al. Beyond semantic organization: Memory as execution state management for long-horizon agents. *arXiv preprint arXiv:2606.06090*, 2026.

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

DeepSeek-AI et al. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Rongfeng Guo, Yinxuan Huang, Yusen Wu, Maoqing Zhong, Yunlu Chen, Meng Tang, Teng Long, and Vincent Tao Hu. From state to action: Ooda-tool for reliable multi-turn tool use. *arXiv preprint arXiv:2608.24368*, 2026.

Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, et al. Memoryarena: Benchmarking agent memory in interdependent multi-session agentic tasks. *arXiv preprint arXiv:2602.16313*, 2026.

Shuo Ji, Yibo Li, and Bryan Hooi. Memory is reconstructed, not retrieved: Graph memory for llm agents. *arXiv preprint arXiv:2606.06036*, 2026.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

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.

Hyunji Lee, Justin Chih-Yao Chen, Joykirat Singh, Zaid Khan, Elias Stengel-Eskin, and Mohit Bansal. Minteval: Evaluating memory under multi-target interference in long-horizon agent systems. *arXiv preprint arXiv:2605.18565*, 2026.

Lizi Liao, Le Hong Long, Yunshan Ma, Wenqiang Lei, and Tat-Seng Chua. Dialogue state tracking with incremental reasoning. *Transactions of the Association for Computational Linguistics*, 9:557–569, 2021.

OpenAI. GPT-5.6 system card. OpenAI Deployment Safety Hub, 2026. URL https://deploymentsafety.openai.com/gpt-5-6.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Zhiming Pan, Junyu Luo, Zhiping Xiao, Kaize Ding, Xiao Luo, and Ming Zhang. Zipact: Zipping interaction history into a compact state for efficient llm agents. *Transactions on Machine Learning Research*, 2026.

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. Qwen Blog, 2026. URL https://qwen.ai/blog?id=qwen3.5.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 2009. doi: 10.1561/1500000019.

Nikolai Rozanov and Marek Rei. Stateact: Enhancing llm base agents via self-prompting and state-tracking. In *Proceedings of the 1st Workshop for Research on Agent Language Models (REALM 2025)*, pp. 367–385, 2025.

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.

Joykirat Singh, Zaid Khan, Archiki Prasad, Justin Chih-Yao Chen, Akshay Nambi, Hyunji Lee, Elias Stengel-Eskin, and Mohit Bansal. Agent-brace: Decoupling beliefs from actions in long-horizon tasks via verbalized state uncertainty. *arXiv preprint arXiv:2605.11436*, 2026.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, 2026.

Md Nayem Uddin, Amir Saeidi, Eduardo Blanco, and Chitta Baral. Ledgeragent: Structured state for policy-adherent tool-calling agents. *arXiv preprint arXiv:2606.20529*, 2026.

Tong Wang, Pei Xu, Shiyue Cao, Likun Yang, Daipeng Li, Jianbin Jiao, and Kaiqi Huang. Samem: State-aware memory as a fine-grained memory for llm agents in decision-making. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 14691–14710, 2026.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

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

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.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026.

A Related Work

A.1 Execution State Tracking for Long-Horizon Agents

In partially observable sequential decision-making, a belief state summarizes the action–observation history into the information needed for current decisions and is recursively updated based on the action taken and the new observation (Kaelbling et al., 1998). In task-oriented dialogue, Dialogue State Tracking updates user goals and relevant slot information as the conversation progresses. The incremental reasoning method of Liao et al. further combines the existing state with information from the current turn to progressively revise the estimate of the user's goals (Liao et al., 2021). Recent LLM agent methods explicitly maintain a task state or belief state that evolves during execution, organizing accumulated facts, constraints, and environmental information into a state representation for decision-making (Rozanov & Rei, 2025; Singh et al., 2026; Uddin et al., 2026; Guo et al., 2026). ZipAct compresses the growing interaction history into a structured state comprising a Goal State, World State, and Constraint State, and decouples state updating from action generation, allowing the agent to act on the current state rather than the full history (Pan et al., 2026). These methods establish the value of maintaining a compact execution state. In long-horizon tasks, however, a further question is how an agent can revisit earlier states and their supporting evidence when a shift in task focus makes them relevant again.

A.2 Long-Term Memory for LLM Agents

Long-term memory methods store historical information in external memory so that information beyond the current context remains accessible in later interactions. Early work explored the preservation and reuse of historical information by accumulating observations and experiences, generating reusable reflections, or managing conversation history through tiered memory (Park et al., 2023; Shinn et al., 2023; Packer et al., 2023). More recent methods improve the organization and updating of historical information through memory consolidation, dynamic linking, graph structures, and temporal relations (Chhikara et al., 2025; Xu et al., 2026; Rasmussen et al., 2025). In extended interactions involving multiple related tasks, agents must also use earlier actions and environmental feedback to guide later decisions. MemoryArena evaluates this need for memory in interdependent multi-session agentic tasks (He et al., 2026). As historical information accumulates and changes, its use presents further challenges: MINTEval reveals multi-target interference under frequent updates (Lee et al., 2026), while STALE shows that, even when updated information is available, agents may fail to recognize that an older memory is no longer valid and adjust their behavior accordingly (Chao et al., 2026). Together, these works highlight the importance of not only retaining historical information, but also selecting relevant memories and assessing their continued applicability as tasks and environments evolve.

A.3 Connecting Execution State and Memory Access

Recent work explores how accumulated experience can support an agent's ongoing decisions. Dynamic Cheatsheet maintains an evolving memory of strategies and problem-solving insights for reuse at inference time (Suzgun et al., 2026). Agentic Context Engineering (ACE) develops this direction by organizing context as an evolving playbook, accumulating and refining strategies through generation, reflection, and curation (Zhang et al., 2026). Other approaches tie memory access more directly to the current decision context: SAMem retrieves state-aligned experiential memories (Wang et al., 2026), while MRAgent uses intermediate evidence to iteratively expand or prune access paths in an associative memory graph (Ji et al., 2026). Together, these methods connect past experience with current decisions through reusable guidance and context-sensitive memory access.

Concurrent work, MAGE (Chen et al., 2026), studies agent memory from the perspective of execution-state management, organizing action–observation trajectories into a hierarchical state tree, constructing working context along the active path, and supporting revision through branching. FlowState instead unifies current execution and cross-request memory through persistent, semantically typed state nodes. Nodes are individually addressable and retain relations and references to raw tool observations, separating persistent retention from visibility in the Active State. PSA discloses historical states and supporting evidence on demand, while ISU creates and revises current-request

Table 5: FlowState improves task performance and reduces token usage across three base models.

	Web Shopping			Math			Physics		
Method	SR ↑	PS ↑	Tok. (M) ↓	SR ↑	PS ↑	Tok. (M) ↓	SR ↑	PS ↑	Tok. (M) ↓
Qwen3.5-397B-A17B									
Full Context	0.00	19.56	134.82	30.00	35.27	19.97	50.00	55.56	2.02
FlowState	3.33	37.22	57.68	37.50	53.96	14.53	60.00	59.72	2.01
Δ (pp / %)	+3.33	+17.66	**-57.22%**	+7.50	+18.69	**-27.24%**	+10.00	+4.16	**-0.50%**
DeepSeek-V4-Flash									
Full Context	0.00	21.11	52.65	30.00	50.36	40.60	55.00	72.50	3.68
FlowState	5.33	42.89	44.83	37.50	55.53	10.96	60.00	73.96	1.78
Δ (pp / %)	+5.33	+21.78	**-14.86%**	+7.50	+5.17	**-73.01%**	+5.00	+1.46	**-51.69%**
GPT-5.6-terra									
Full Context	0.00	33.89	313.95	42.50	54.23	14.59	50.00	62.60	2.76
FlowState	4.00	51.44	49.29	45.00	59.25	8.28	50.00	67.65	1.62
Δ (pp / %)	+4.00	+17.55	**-84.30%**	+2.50	+5.02	**-43.26%**	0.00	+5.05	**-41.33%**

Note. SR and PS are percentages; token counts are in millions (M). Δ compares FlowState with Full Context: SR/PS changes are in percentage points (pp), while token changes are relative percentages calculated from unrounded counts. Negative token changes indicate savings.

states informed by the disclosed information. Together, these mechanisms enable the same state representation to support current decisions and cross-request reuse.

B Supplementary Materials

B.1 Performance and Efficiency Gains Across Base Models

As shown in Table 5, FlowState consistently improves the performance and token efficiency of Qwen3.5-397B-A17B, DeepSeek-V4-Flash, and GPT-5.6-terra. Across all nine model–task combinations, it improves PS and reduces token consumption, while increasing SR in eight cases and preserving it in the remaining one.

The gains are particularly pronounced on Web Shopping, where all three models improve PS by 17.55–21.78 percentage points and achieve nonzero SR from zero-success baselines. These performance gains are accompanied by substantial token savings in several settings. GPT-5.6-terra reduces token consumption by 84.30% on Web Shopping, while DeepSeek-V4-Flash reduces token consumption by 73.01% on Math and improves SR by 7.50 percentage points. Improvements also occur when baseline performance is already relatively high. On Physics, DeepSeek-V4-Flash increases PS from 72.50% to 73.96% while reducing token consumption by 51.69%.

These results show that FlowState delivers consistent benefits across the evaluated models and tasks, combining improved task performance with lower token consumption even when baseline performance is already relatively high.

B.2 State Types and Semantic Scope

Table 6 summarizes the five state types and their semantic scope. These types organize information by its source and role in execution, distinguishing user requirements, environmental observations, and agent-generated assessments and outputs. The same taxonomy applies to both newly created states and historical states accessed during subsequent requests.

Table 6: State types and their semantic scope.

Perspective	State Type	Semantic Scope
User	preferences	User preferences, constraints, and requirements extracted from requests.
Environment	knowledge	Task-relevant observations recorded from tool outputs.
Environment	attributes	Entity-centric information distilled from tool outputs, including properties and characteristics.
Agent	judgements	Agent assessments of task goals, plans, intermediate decisions, and execution risks.
Agent	artifacts	Reusable artifacts, tools, code, and other outputs created by the agent during task execution.

B.3 Runtime Prompt Templates

The templates below show the task-agent prompt structure for MemoryArena and τ^3-Bench. Each method has separate system and user messages. Braced fields are populated at runtime; ReAct rules are shared, while policy and state rules vary by task and state algorithm, respectively. Output protocols place method-specific state fields before `tool_calls` and `assistant_message`. Ellipses in the FlowState status denote additional query or state entries. Colored left rules mark the FlowState-specific instructions, output protocol, and state input.

FlowState

System Instruction

```
<instructions>
{react_rules}
## flowstate_rules
Use the Active State block in <agent_status> as your working memory.
Before each action, check whether it contains the facts and identifiers
required for that action.
### Incremental State Update
- Update Active State whenever task-relevant facts, requirements, plans,
or results change.  Output only the changes; do not regenerate the
entire state.
- Assign each state to preference (user requirements and constraints),
knowledge (tool observations), attribute (entity properties), judgement
(goals, plans, decisions, and risks), or artifacts (reusable outputs).
- Apply state_delta to the Active State block:
  - add:  Insert a new state with a new id.
  - update:  Revise an existing Active State entry by its id.
  - remove:  Delete an existing Active State entry by its id.
- Do not update or remove entries whose ids appear in the Historical
State Index, even if their full contents have been expanded into Active
State.  To revise historical information, add a new Active State entry
and link it to the historical state through refs.
- Give each state a semantic id, a concise summary, and a structured
value.  Link it to supporting evidence and related states through refs.
- Declare directed relations through refs.  Every edge points from the
state being added or updated (source) to the state_id in that ref
(target).  All relation types use this direction.  Set rel to one of:
supports, derives, follows, supersedes.
### Progressive State Access
- If required information is missing, use prior queries and state
summaries in the Historical State Index, together with related_states in
Active State, to select relevant historical content.  Use the
corresponding state_id as state_focus.target_id.
- Specify what information you need and which decision or action it
enables.  Never infer exact facts or identifiers from summaries.
- A related_states entry identifies a related state but does not reveal
its full content.  Request that state separately when its details are
needed.
```

```
- Use newly disclosed content only from the next model invocation
onward.  Defer any action that depends on it.  Do not request content
already visible.
</instructions>
<policy>
{task_policy}
</policy>
<tools>
{available_tools}
</tools>
<output_protocol>
Return exactly four blocks in the following order.  Use valid JSON
inside each block.  Do not include code fences or text outside the
blocks.
```text
<state_focus>
[{"target_id":  "<visible_id>", "reason":  "<missing information and
intended use>"}]
</state_focus>
<state_delta>
[{"state_set":  "<state_type>", "operation":  "add", "state_id":
"<state_id>", "summary":  "<concise summary>", "value":  {}, "refs":
[{"rel":  "<relation>", "state_id":  "<source_state_id>"}]}]
</state_delta>
<tool_calls>
[{"name":  "<tool_name>", "arguments":  {}}]
</tool_calls>
<assistant_message>
{"content":  null}
</assistant_message>
```
Set operation to add, update, or remove. For an execution decision,
request a tool call or provide a user-facing message.
At each step, jointly propose state updates, historical access, and an
execution decision whenever the available information supports them. Do
not split independent operations across steps.
Request missing historical information and perform any independent
operations in the same step. Wait for the requested information before
making updates or taking actions that require it.
Always output all four protocol blocks. Use [] for unused arrays and
set assistant_message.content to null when no user-facing reply is
needed. At least one operation must be non-empty.
</output_protocol>
```

Input Prompt

```
<agent_status>
Historical State Index:
[
  {
    "user_query":  "{prior_query}",
    "query_id":  {prior_query_id},
    "states":  [
      {
        "state_id":  "{historical_state_id}",
        "summary":  "{historical_state_summary}"
      }
    ]
  },
  ...
]
Active State:
[
  {
    "state_id":  "{active_state_id}",
```

```
    "type":  "{state_type}",
    "related_states":  [
      {"state_id":"{related_state_id}","rel":"{relation}"}
    ],
    "summary":  "{active_state_summary}",
    "value":  {state_value_json}
  },
  ...
]
</agent_status>
<latest_observation>
{observation}
</latest_observation>
<user_query>
{query}
</user_query>
```

Full Context

System Instruction

```
<instructions>
{react_rules}
</instructions>
<policy>
{task_policy}
</policy>
<tools>
{available_tools}
</tools>
<output_protocol>
{react_output_protocol}
</output_protocol>
```

Input Prompt

```
<trajectory_history>
{full_trajectory}
</trajectory_history>
<latest_observation>
{observation}
</latest_observation>
<user_query>
{query}
</user_query>
```

Memory System

System Instruction

```
<instructions>
{react_rules}
</instructions>
<policy>
{task_policy}
</policy>
<tools>
{available_tools}
</tools>
<output_protocol>
{react_output_protocol}
</output_protocol>
```

Input Prompt

```
<memory>
{retrieved_memory}
</memory>
```

```
<trajectory_history>
{query_trajectory}
</trajectory_history>
<latest_observation>
{observation}
</latest_observation>
<user_query>
{query}
</user_query>
```

State Tracking

System Instruction

```
<instructions>
{react_rules}
{state_tracking_rules}
</instructions>
<policy>
{task_policy}
</policy>
<tools>
{available_tools}
</tools>
<output_protocol>
{state_output_protocol}
</output_protocol>
```

Input Prompt

```
<agent_status>
{execution_state}
</agent_status>
<latest_observation>
{observation}
</latest_observation>
<user_query>
{query}
</user_query>
```

State Tracking + Memory

System Instruction

```
<instructions>
{react_rules}
{state_tracking_rules}
</instructions>
<policy>
{task_policy}
</policy>
<tools>
{available_tools}
</tools>
<output_protocol>
{state_output_protocol}
</output_protocol>
```

Input Prompt

```
<memory>
{retrieved_memory}
</memory>
<agent_status>
{execution_state}
</agent_status>
<latest_observation>
{observation}
</latest_observation>
```

```
<user_query>
{query}
</user_query>
```

B.4 Execution Rules

For request q_k at decision step t, the execution rules P define the admissibility conditions for the model-proposed state delta $\Delta S_{k,t}$ and access set $R_{k,t}$. The runtime applies these conditions through $\mathrm{Validate}_{\mathrm{upd}}$ and $\mathrm{Validate}_{\mathrm{acc}}$, yielding the validated requests $\widehat{\Delta S}_{k,t}$ and $\widehat{R}_{k,t}$, respectively.

Incremental State Update. Each Add operation in $\Delta S_{k,t}$ requires an unused semantic identifier. The target of each Update or Remove operation must be present in $S_{k,t}$ and must have been created during the current request. References in proposed state relations must resolve to valid node identifiers. Operations failing these conditions are excluded from $\widehat{\Delta S}_{k,t}$.

Progressive State Access. Let $\mathrm{IDs}(G_k)$ denote the historical state identifiers in G_k, and let $\mathrm{ToolRefs}(S_{k,t})$ denote the identifiers of raw tool results referenced by states in the current Active State $S_{k,t}$. The validated access set is restricted to

$$\widehat{R}_{k,t} \subseteq R_{k,t} \cap \big(\mathrm{IDs}(G_k) \cup \mathrm{ToolRefs}(S_{k,t})\big). \tag{5}$$

Requested identifiers outside this set are rejected by $\mathrm{Validate}_{\mathrm{acc}}$ and excluded from $\widehat{R}_{k,t}$.

B.5 Experimental and Implementation Details

Benchmarks and evaluation scope. MemoryArena tests continual reuse of historical information across interdependent subtasks. We evaluate a subset of its environments: Bundled Web Shopping requires compatibility-aware purchases across sessions; Group Travel Planning introduces travelers and interdependent preferences over time; and Formal Reasoning (Math and Physics) requires reusing intermediate results in later derivations. In τ^3-Bench, Airline and Retail involve policy-constrained interactions with simulated users, including information lookup, flight rebooking, and refunds.

We exclude Progressive Web Search because we identified issues with its ground-truth annotations that undermine the reliability of answer-based evaluation. To avoid confounding method comparisons with annotation errors, we omit this environment from all reported results and aggregate metrics.

The evaluated MemoryArena task sets comprise 150 shopping tasks (900 subtasks), 270 travel-planning tasks (1,869 subtasks), 40 math tasks (354 subtasks), and 20 physics tasks (86 subtasks). For τ^3-Bench, we use the base task sets of 50 airline tasks and 114 retail tasks. The evaluation across base models uses the Web Shopping, Math, and Physics task sets, while the component ablation uses the Web Shopping and Travel Planning task sets. The analysis of performance across subtask depths reuses the main evaluation data and includes only tasks containing the corresponding subtask.

Evaluation metrics. On MemoryArena, SR requires a final bundle or plan satisfying all group members in Shopping and Travel, or a correct final subtask in Math and Physics. PS averages the fraction of correctly completed subtasks per task for Shopping, Math, and Physics; for Travel, it is the fraction of correctly completed subtasks among all evaluated subtasks. sPS credits satisfied constraints within each Travel subtask. On τ^3-Bench, Pass Rate measures task success, DB Accuracy the correctness of the final database state, and Action Accuracy measures reference-action matching, not overall action correctness. #Tok includes prompt and generation tokens.

Model configurations. We evaluate FlowState with three backbone models: DeepSeek-V4-Flash, GPT-5.6-terra, and Qwen3.5-397B-A17B. The full-context baseline is evaluated with all three models, whereas the memory-system, state-tracking, and hybrid baselines use DeepSeek-V4-Flash. For agent inference, we specify the vLLM request parameters `temperature=0.1`, `top_p=1.0`, and `max_completion_tokens=32768`, with thinking disabled (`enable_thinking=false`). No other generation parameters are overridden.

Benchmark protocol and prompt construction. We use the MemoryArena and τ^3-Bench implementations at the revisions specified below.[1] For the domains in Section 3.1, we retain the benchmark task configurations, environment tools and their descriptions, and evaluation procedures. We adapt the task-agent prompt assembly to the system- and user-message structures in Appendix B.3, combining benchmark task policies and tool descriptions with method-specific instructions and context. All methods compared within a domain use the same benchmark tasks, environment tools, and evaluation procedures.

Baselines. Full Context appends the available interaction history to the agent context without explicit state abstraction or retrieval. ReasoningBank, BM25, and Mem0 store and retrieve historical information. Our ReasoningBank and BM25 baselines follow the implementations distributed with MemoryArena; we reproduce Mem0 from its publicly released code.[2] We retain the released memory-agent prompts for ReasoningBank and Mem0. ZipAct maintains a compact execution state and is reproduced from its publicly released code.[3] For ZipAct, we populate the domain-specific injectables in its state-construction prompts with state items relevant to each benchmark task, as specified by Pan et al. (2026); the remaining state-module prompt content follows the released implementation. ZipAct+BM25 augments ZipAct with BM25 retrieval over the interaction history.

Ablation Design and Scope. To evaluate the roles of ISU and PSA in the full framework, we construct each ablation variant by replacing one component with an alternative mechanism while preserving the functionality of the other. Specifically, **w/o PSA** retains ISU and directly includes the full contents of all historical states in the Active State, comparing full exposure of historical states with on-demand access. Conversely, **w/o ISU** retains PSA and constructs a state node from the final response at the end of each query, linking it to the raw tool results from that query through fixed rules without involving ISU. This variant still supports on-demand access to historical final responses and their associated tool results. Thus, the w/o ISU ablation compares incremental state maintenance during execution with final-response-based state construction at query boundaries, rather than the presence or absence of access to historical information or evidence. These ablations assess the overall contribution of each mechanism without isolating the contributions of its individual design choices.

[1] MemoryArena revision: `6cd9de1`; τ^3-Bench v1.0.1: `fc0055d`.

[2] Mem0 revision: `8d6c001`.

[3] ZipAct revision: `f025804`.