TokenPilot: Cache-Efficient Context Management for LLM Agents

Buqiang Xu[1]*, Zirui Xue[1]*, Dianmou Chen[2]*, Chenyang Fu[3]*, Chiyu Wu[4]*, Caiying Huang[3]*, Chen Jiang[1], Jizhan Fang[1], Xinle Deng[1], Yijun Chen[1], Yunzhi Yao[1], Xuehai Wang[1], Jin Shang[4], Gong Yu[4], Ningyu Zhang[1]†

[1]Zhejiang University [2]University of Electronic Science and Technology of China
[3]Xi'an University of Electronic Science and Technology [4]HomologyAI

Abstract

As LLM agents are deployed in long-horizon sessions, context accumulation drives up inference costs. Existing approaches utilize text pruning or dynamic memory eviction to minimize token footprints; however, their unconstrained sequence mutations alter layouts, introducing prefix mismatches and cache invalidation. This reveals a critical trade-off between text sparsity and prompt cache continuity. To address this, we present **TokenPilot**, a dual-granularity context management framework. Globally, *Ingestion-Aware Compaction* acts as a framework harness to stabilize prompt prefixes and eliminate open-world environmental noise at the ingestion gate. Locally, *Lifecycle-Aware Eviction* monitors the ongoing residual utility of context segments, enforcing a conservative batch-turn schedule to offload content segments only when task relevance expires. Experiments on `PinchBench` and `Claw-Eval` under both isolated and continuous modes demonstrate that TokenPilot reduces costs by **61%** and **56%** in isolated mode, and **61%** and **87%** in continuous mode, while maintaining competitive performance compared to prior systems[1].

1 Introduction

The paradigm of large language models has shifted from conversational assistants (Ouyang et al., 2022) to stateful execution controllers (Anthropic, 2025; OpenAI, 2026; OpenClaw, 2026) orchestrating complex tools (Li et al., 2025a; Merrill et al., 2026), file systems (Jimenez et al., 2024), and cross-application workflows (Zhou et al., 2024). Consequently, the core challenge of agent design has transitioned to real-world operational reliability (Ye et al., 2026; Kilo AI Team, 2026). However, continuous multi-turn interactions inevitably accumulate verbose execution traces, rapidly inflating sequence lengths and escalating per-turn inference costs. Managing this context growth is thus an essential prerequisite for sustainable real-world deployment (Hu et al., 2025b; Mei et al., 2025).

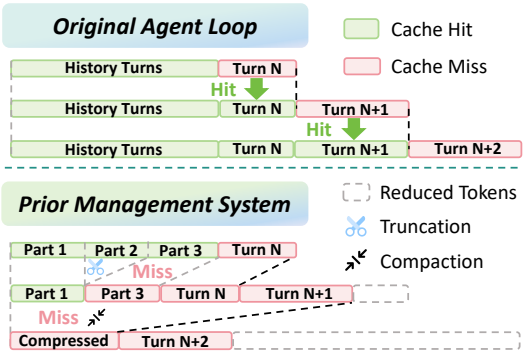


Figure 1: Comparison of cache alignment behaviors. While the *Original Agent Loop* maintains continuous layouts to achieve cumulative **cache hits**, previous management systems execute text truncation or compaction that mutates input boundaries, inadvertently triggering severe backend KV **cache misses**.

The research community has primarily approached this challenge from a content-reduction perspective. Initial efforts focus on static text compression, pruning low-utility tokens (Jiang et al., 2023; Pan et al., 2024) or sentences (Li et al., 2023) before prompt transmission. For dynamic execution traces, existing architectures implement context folding (Ye et al., 2025b; Feng et al., 2026), or demand paging (Mason, 2026) to condense intermediate reasoning backbones (Qian et al., 2026) and offload continuous trajectories to external storage (Li et al., 2025b; Hu et al., 2026).

Despite their success in textual compaction, these methods introduce a fundamental trade-off between prompt reduction and hardware cache efficiency (Kwon et al., 2023; Zheng et al., 2024). As shown in Figure 1, while aggressively truncating or shifting context pages minimizes per-turn token counts, this constant layout mutation shatters prompt prefix continuity. The resulting hard-

* Equal contribution.
† Corresponding author.
[1]TokenPilot has been integrated into LightRSI at `https://github.com/zjunlp/LightRSI`.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 19034–19049
October 24-29, 2026 ©2026 Association for Computational Linguistics

ware pre-fill penalties and cache invalidations ultimately override any financial savings from text reduction. **We argue that an effective framework must fundamentally reconcile text-level sparsity with hardware cache alignment.** To achieve this design synergy, the deployment system must simultaneously safeguard physical prefix continuity during observation ingestion and defer structural memory eviction until a trajectory's residual utility thoroughly expires.

Building on this insight, we present **TokenPilot**, a dual-granularity context management framework that reconciles sequence reduction with prompt cache alignment. At the global level, *Ingestion-Aware Compaction* acts as a deterministic harness to optimize the layout at the initial warm-up phase rather than retroactively compressing an existing cache. Specifically, it neutralizes volatile runtime variables via stable placeholders and shifts tool definitions downstream to secure a byte-identical prompt prefix from the first turn, while concurrently stripping structural noise from incoming tool responses before ingestion. At the local level, *Lifecycle-Aware Eviction* monitors active execution trajectories online by evaluating their dynamic residual utility. Rather than executing frequent, disruptive memory paging, the eviction pass remains strictly conservative, deferring structural purge until the segment's residual value thoroughly expires to safeguard context continuity.

Evaluated on `PinchBench` and `Claw-Eval` under commercial pricing structures, TokenPilot dramatically reduces total inference monetary expenditures by **61%** and **56%** in isolated mode, and **61%** and **87%** in continuous mode, while successfully maintaining competitive task performance.

2 Background

Task Settings. We consider an agent processing a sequence of tasks $\mathcal{S} = \{t_1, t_2, \ldots, t_n\}$. Each task generates a trajectory of instructions, reasoning traces, tool calls, and responses, which accumulate into the session context $\mathcal{C}$. We evaluate under two modes: *isolated mode*, where the context $\mathcal{C}$ is reset at each task boundary, and *continuous mode*, where histories persist across the entire sequence.

Optimization Objective. A context management framework $\mathcal{M}$ transforms the raw history $\mathcal{C}$ into an optimized runtime context $\mathcal{C}' = \mathcal{M}(\mathcal{C})$. The objective is to maximize the ratio of context utility to maintenance cost:

$$\max_{\mathcal{M}} \frac{\sum_{m \in \mathcal{C}'} \hat{U}(m \mid \mathcal{C}')}{\mathcal{K}(\mathcal{C}')} \tag{1}$$

Here, context utility quantifies the necessity of context tokens for guiding downstream reasoning and tool execution, where $\hat{U}(m \mid \mathcal{C}')$ estimates the marginal contribution of message m to subsequent agent actions. The serving cost $\mathcal{K}(\mathcal{C}')$ is governed by the backend KV prompt-caching mechanism:

$$\mathcal{K}(\mathcal{C}') = \alpha \cdot |\mathcal{C}'_{\text{hit}}| + |\mathcal{C}'_{\text{miss}}| \tag{2}$$

where $\mathcal{C}'_{\text{hit}}$ and $\mathcal{C}'_{\text{miss}}$ denote tokens served from the cache at a discounted cost rate $\alpha \ll 1$ and those incurring full pre-fill cost, respectively, subject to the length alignment constraint $|\mathcal{C}'| = |\mathcal{C}'_{\text{hit}}| + |\mathcal{C}'_{\text{miss}}|$.

3 TokenPilot

3.1 Overall

We propose **TokenPilot**, a dual-granularity framework that addresses the context optimization objective across two complementary operational levels. At the *global* framework level, *Ingestion-Aware Compaction* (§3.2) acts as a deterministic harness at the ingestion boundary, standardizing layouts and purifying incoming messages to optimize the sequence during the initial cache warm-up phase. At the *local* sequence level, *Lifecycle-Aware Eviction* (§3.3) dynamically monitors the residual utility of active trajectories, enforcing a conservative batch-turn schedule to purge segments only when their task-level utility has thoroughly expired.

3.2 Global Ingestion-Aware Compaction

Ingestion-Aware Compaction acts as a framework harness to optimize sequence layout at the ingestion boundary. Based on the interaction loop, we partition the message space into two functional categories. Let Ω_{int} denote *internal intentional messages* generated natively by the system or model, encompassing task prompts, thinking traces, tool calls, and final responses, which naturally possess high intrinsic utility density. Conversely, let Ω_{env} denote *open-world environmental feedback* from unmanaged tools, which inherently suffer from extensive structural clutter. The marginal utility density for an incoming message m is formalized as:

$$\hat{U}(m) = \begin{cases} 1 & m \in \Omega_{\text{int}} \\ \max(\gamma(m), \mathbb{G}(m)) & m \in \Omega_{\text{env}} \end{cases} \tag{3}$$

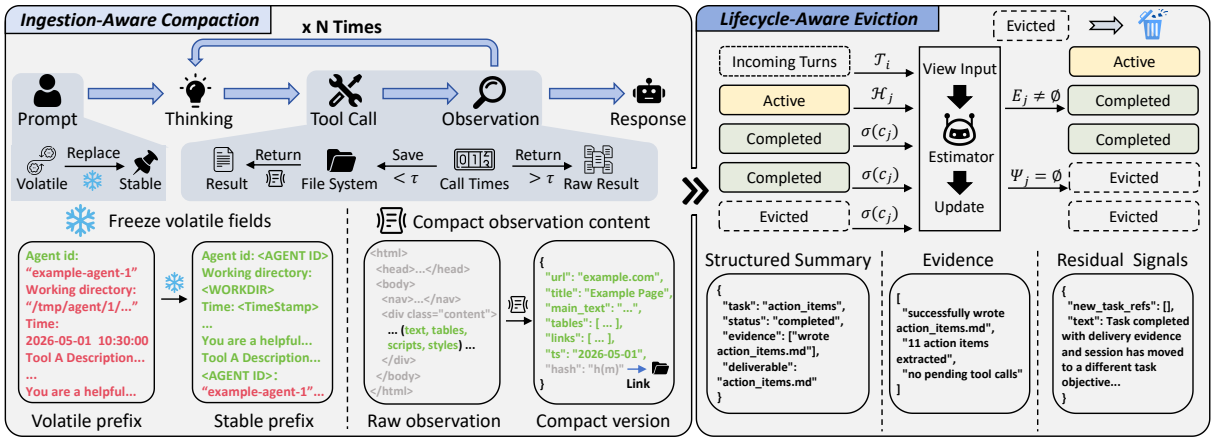


Figure 2: The system architecture of TokenPilot, featuring Ingestion-Aware Compaction at the global framework harness level and Lifecycle-Aware Eviction at the local context sequence level.

where $\gamma(m) \leq 1$ is the base environmental feedback utility density, and $\mathbb{G}(m) = \mathbb{1}[f(h(m)) > \tau]$ is an ingestion gate. When the access frequency $f(h(m))$ of a content hash exceeds threshold τ, $\mathbb{G}(m)$ switches to 1, fully upgrading the density to restore comprehensive content delivery.

Prefix Stabilization. Cross-task KV cache reuse is frequently disrupted by runtime-volatile fields within Ω_{int} that introduce position jitter and prefix mismatches. TokenPilot implements a canonicalization operator to secure a byte-identical prompt prefix from the first turn:

$$\phi(m^{(t)}) = \phi(m^{(t+1)}) \implies \mathcal{C}'_{\text{prefix}} \subseteq \mathcal{C}'_{\text{hit}} \quad (4)$$

where ϕ intercepts internal messages at the harness level and substitutes volatile runtime markers with static placeholders. By preserving physical continuity across tasks, this operator directly eliminates full-cost pre-fill penalties.

Observation Reduction. For environmental messages $m \in \Omega_{\text{env}}$ where $\mathbb{G}(m) = 0$, TokenPilot applies deterministic reduction passes at the ingestion gate to lift token utility. The transformation and its reliable fallback loop are formalized as:

$$m_{\text{ingest}} = \kappa(m), \quad \mathcal{A}[h(m)] \leftarrow m \quad (5)$$

where $\kappa(m)$ denotes the compacted structural preview stored in working memory, and $\mathcal{A}$ represents an external artifact registry indexed by content hash $h(m)$ to ensure total operational safety. If the compressed message lacks critical signals during execution, the agent harness invokes a lightweight recovery tool to dynamically recall the full payload $\mathcal{A}[h(m)]$, automatically upgrading the status to disable subsequent truncation for that path.

3.3 Local Lifecycle-Aware Eviction

At the local sequence level, Lifecycle-Aware Eviction dynamically regulates the historical retention window based on a segment's ongoing task utility. To safeguard physical cache continuity and prevent disruptive turn-by-turn memory paging, TokenPilot tracks each context segment c_j through three progressive states $s_j \in \{active, completed, evictable\}$ maintained in a framework registry $\mathcal{R}$. The marginal utility of a segment is formalized as:

$$\hat{U}(c_j) = \begin{cases} 1 & s_j = active \\ \mathbb{1}[\Psi_j \neq \emptyset] & s_j = completed \\ 0 & s_j = evictable \end{cases} \quad (6)$$

where Ψ_j represents the quantified *residual utility* of the segment. Under this formulation, a segment that has concluded its execution path is not immediately truncated; it transitions to a conservative *completed* state, retaining its physical cache slots as long as its residual relevance to ongoing interactions remains non-zero ($\Psi_j \neq \emptyset$).

Context State Estimation and Execution. To suppress spurious state transitions, an online model-based estimator $\mathcal{E}$ is triggered conservatively in stable batches of B turns rather than at every execution step. For the i-th batch, the estimator ingests a compressed historical view $\mathcal{V}_i$ to compute state updates over each segment:

$$\Delta\mathcal{R}_i^{(j)} = \langle E_j, \Psi_j \rangle = \mathcal{E}(\mathcal{V}_i, \mathcal{R}_{i-1}) \quad (7)$$

where E_j denotes explicit resolution evidence showing the sub-task has achieved its objective, and Ψ_j represents residual utility signals extracted

Method	Overall ↑	Performance by Category ↑											Input Tokens (M)		Output (M)	Cost ($) ↓
		Prod	Res	Write	Code	Anal	CSV	Log	Meet	Mem	Skill	Integ	Cache Hit	Cache Miss		
Isolated Mode																
Vanilla	80.5	87.2	68.7	84.1	86.0	75.1	83.0	94.7	81.4	86.5	70.3	55.3	6.184	8.753	0.285	8.31
LLMLingua-2	76.9	89.3	64.0	82.1	86.9	80.8	79.6	84.4	66.3	85.0	79.6	72.1	14.241	3.975	0.384	5.78
SelectiveContext	76.5	88.5	64.5	73.0	83.7	82.6	81.1	92.8	63.3	86.9	82.8	77.2	11.273	4.642	0.324	5.79
LCM	77.8	90.1	64.9	79.6	85.4	81.3	81.0	87.1	67.5	85.0	81.7	80.6	16.018	3.064	0.356	5.10
Pichay	78.9	85.4	58.9	71.8	79.0	88.3	79.8	83.6	84.0	91.3	69.8	63.3	6.717	3.333	0.238	4.07
Summary	79.5	80.7	66.3	83.5	77.9	82.1	87.5	77.2	81.3	92.5	67.2	54.4	12.303	3.009	0.296	4.51
MemoBrain	78.1	86.8	62.1	88.9	85.7	82.6	88.3	85.4	63.6	92.5	76.1	69.7	10.200	2.107	0.233	3.36
AgentSwing	78.4	89.8	71.9	80.2	79.5	83.5	80.8	83.7	77.9	92.5	65.7	35.0	4.534	7.129	0.241	6.77
Keep-Last-N	80.4	86.0	70.0	82.4	80.1	77.6	78.3	91.5	84.3	92.5	70.1	87.8	12.813	2.657	0.291	4.26
MemOS	79.4	84.2	54.4	83.1	82.3	78.2	81.1	97.2	77.6	92.5	85.9	80.2	29.018	4.573	0.492	7.81
TokenPilot	**81.0**	89.0	71.2	80.0	72.6	88.9	85.3	95.2	79.4	95.0	95.2	58.0	8.893	1.933	0.244	**3.22**
Continuous Mode																
Vanilla	79.2	83.5	58.4	86.8	80.0	78.5	87.8	94.6	77.6	95.0	55.8	83.6	25.015	5.943	0.202	7.24
LLMLingua-2	73.8	85.8	58.4	80.3	74.3	79.6	82.8	84.2	63.4	90.0	79.1	83.6	20.574	2.183	0.194	4.06
SelectiveContext	74.0	85.4	64.2	83.1	75.4	78.8	77.3	91.2	62.2	89.5	71.0	80.3	25.475	2.608	0.196	4.75
LCM	77.0	88.1	63.2	90.1	75.7	78.5	85.4	88.9	65.1	82.8	80.8	78.2	18.708	2.417	0.222	4.21
Pichay	76.5	88.0	66.7	76.2	81.0	77.6	83.5	84.2	67.6	100.0	63.8	75.3	11.698	6.874	0.260	7.20
Summary	78.4	89.1	64.4	73.8	82.9	69.6	81.6	93.6	80.3	95.0	61.7	75.3	20.687	6.249	0.196	7.12
MemoBrain	78.0	87.7	65.0	85.5	84.9	75.9	81.0	89.0	72.3	90.3	86.6	84.7	12.917	2.283	0.232	3.73
AgentSwing	78.5	86.3	67.3	89.0	79.1	82.4	87.4	68.1	72.4	93.8	61.7	83.8	12.680	5.476	0.314	6.47
Keep-Last-N	79.1	86.3	67.0	87.8	87.0	77.0	85.4	77.3	75.9	95.0	56.8	75.1	18.117	4.481	0.209	5.66
MemOS	80.9	87.5	59.0	85.4	87.1	82.0	81.0	95.0	78.1	92.5	87.4	84.1	30.859	8.939	0.308	10.41
TokenPilot	**81.3**	76.7	76.9	90.6	84.1	86.0	85.6	89.1	73.6	95.0	77.2	80.1	8.551	1.549	0.219	**2.79**

Table 1: Performance and resource consumption comparison on PinchBench under isolated and continuous modes. ↑: larger is better; ↓: smaller is better. **Best results** in bold, second-best underlined. **Input Tokens** are decomposed into **Cache Hit** and **Cache Miss** tokens, reflecting prefix stability and reuse efficiency. Category abbreviations: Prod=Productivity, Res=Research, Write=Writing, Code=Coding, Anal=Analysis, CSV=CSV Analysis, Log=Log Analysis, Meet=Meeting Analysis, Mem=Memory, Skill=Skills, Integ=Integrations.

from dependency patterns. TokenPilot enforces a gated pipeline to transition these lifecycle states:

$$active \xrightarrow{E_j \neq \emptyset} completed \xrightarrow{\Psi_j = \emptyset} evictable \quad (8)$$

The registry executes strict system validation, updating via $\mathcal{R}_i \leftarrow \mathcal{R}_{i-1} \oplus \Delta\mathcal{R}_i$ only for valid transitions. Once a segment drops to $s_j = evictable$, its utility decays to zero, and the framework executes a single-pass structural purge to construct the optimized context window $\mathcal{C}'$:

$$\mathcal{C}' = \{m \in \mathcal{C} \mid s_{j(m)} \neq evictable\} \quad (9)$$

where $j(m)$ maps message m to its segment index. This batch-gated execution guarantees that eviction remains highly restrained, maximizing cache continuity by eliminating volatile text mutation.

To operationalize this, the estimator $\mathcal{E}$ is instantiated via `Qwen3.5-35B-A3B` as a lightweight, zero-shot validator, incurring negligible overhead; for instance, its total operational cost across the continuous PinchBench stream is less than **$0.03**.

4 Experiments

4.1 Experimental Setup

Benchmarks and Metrics. We evaluate TokenPilot on PinchBench and `Claw-Eval` across both *isolated* and *continuous* modes (see Appendix A.1 for dataset statistics). We track task accuracy alongside actual monetary expenditures. To ensure empirical fidelity, all cache hit and miss token counts are gathered directly from the explicit metadata fields returned by the provider APIs, eliminating client-side estimation errors. Joint scoring formulas and pricing tiers are detailed in Appendix A.2.

Implementation Details. We compare TokenPilot against compression methods (LLMLingua-2, SelectiveContext, Keep-Last-N) and dynamic paging or summarization approaches (Summary, LCM, Pichay, MemoBrain, AgentSwing, MemOS). All evaluated methods utilize `GPT-5.4-mini` as the agent backbone. Detailed hyperparameter configurations for all baselines are documented in Appendix A.3. The exact execution thresholds, model assignments, and system prompts for `TokenPilot` are detailed in Appendix A.4.

4.2 Overall Performance

Table 1 and Table 2 report the performance and resource consumption on PinchBench and `Claw-Eval` under both evaluation modes.

Isolated Mode. TokenPilot outperforms all evaluated baselines by securing the lowest total inference costs of **$3.22** on PinchBench and **$2.27**

Method	Overall ↑	Performance by Category ↑											Input Tokens (M)		Output (M)	Cost ($) ↓
		Wkfl	Ops	Fin	Off	Comm	Prod	Oprn	Safe	Term	MM	Oth	Cache Hit	Cache Miss		
Isolated Mode																
Vanilla	**64.5**	65.4	70.8	45.7	44.4	73.2	70.9	77.7	74.0	56.8	41.0	69.2	9.429	4.637	0.216	5.16
LLMLingua-2	61.9	58.7	67.5	57.6	43.3	62.9	70.1	62.4	61.0	49.6	44.0	75.2	8.169	4.043	0.182	4.44
SelectiveContext	60.7	59.1	68.2	46.3	36.9	61.5	75.5	59.2	67.2	53.1	44.0	74.7	8.271	3.862	0.181	4.31
LCM	61.2	59.0	67.3	51.1	47.7	65.9	76.6	58.4	58.6	51.4	41.5	72.2	9.776	3.543	0.172	4.17
Pichay	59.3	57.3	62.1	38.2	39.4	68.5	65.0	91.6	64.1	25.6	55.0	76.5	4.648	3.944	0.186	4.14
Summary	62.0	70.0	71.0	32.2	20.6	80.0	68.5	82.8	49.2	20.0	41.0	71.4	2.935	2.871	0.174	3.16
MemoBrain	58.0	64.5	60.5	26.1	37.6	56.1	59.9	71.0	63.4	20.0	41.0	75.3	18.182	5.118	0.332	6.69
AgentSwing	60.9	64.2	66.5	44.1	45.7	67.8	52.8	85.8	57.2	25.6	53.6	68.8	4.580	3.585	0.194	3.91
Keep-Last-N	61.8	67.1	73.8	44.7	21.6	54.5	63.6	86.2	38.4	39.4	55.0	69.1	4.229	1.845	0.186	2.54
MemOS	61.6	64.7	74.2	40.9	25.2	71.2	32.0	73.6	80.2	20.0	56.2	74.6	12.582	2.709	0.363	4.61
TokenPilot	63.1	68.1	75.4	47.0	22.3	71.8	65.0	72.0	47.8	37.0	45.6	69.9	4.436	1.154	0.239	**2.27**
Continuous Mode																
Vanilla	**63.4**	70.8	80.3	26.7	27.8	62.2	73.4	78.4	63.6	20.0	41.0	69.6	709.845	21.981	2.622	81.52
LLMLingua-2	59.0	58.7	71.3	34.8	30.6	61.9	65.3	77.6	64.6	20.0	41.0	72.4	575.654	37.197	2.630	82.91
SelectiveContext	56.5	58.1	71.6	21.8	21.2	54.7	74.0	57.7	66.4	20.0	41.0	72.3	437.114	48.678	2.754	81.69
LCM	61.4	66.8	69.0	38.3	29.5	63.3	74.9	66.6	67.3	20.0	41.0	72.7	383.007	28.714	2.691	62.37
Pichay	61.0	69.5	63.8	40.3	24.0	63.1	67.0	94.1	52.5	21.6	41.0	71.0	97.431	63.510	1.046	59.65
Summary	61.6	63.6	74.5	35.3	20.6	55.5	70.1	87.1	66.1	69.0	42.6	66.9	59.772	10.143	1.001	16.59
MemoBrain	57.9	65.9	55.0	24.9	36.7	47.8	73.5	64.2	60.6	20.0	38.4	81.6	47.497	13.990	1.134	19.16
AgentSwing	62.2	67.6	66.5	48.6	36.8	70.0	63.8	90.7	31.7	22.4	41.0	72.8	53.776	10.027	0.907	15.63
Keep-Last-N	60.7	65.3	74.0	35.5	20.8	54.1	73.6	91.9	35.7	59.5	42.4	64.7	44.812	9.106	0.780	13.70
MemOS	57.7	55.9	65.0	56.3	22.2	44.8	64.6	68.8	89.0	20.0	39.6	71.5	49.742	25.432	0.293	24.12
TokenPilot	60.8	58.8	61.8	52.5	32.1	64.2	57.3	89.2	65.8	76.8	45.2	70.9	21.430	9.928	0.338	**10.58**

Table 2: Performance and resource consumption comparison on `Claw-Eval` under isolated and continuous modes. ↑: larger is better; ↓: smaller is better. **Best results** in bold, second-best underlined. **Input Tokens** are decomposed into **Cache Hit** and **Cache Miss** tokens. Category abbreviations: Wkfl=Workflow, Ops=Ops, Fin=Finance, Off=Office QA, Comm=Communication, Prod=Productivity, Oprn=Operations, Safe=Safety, Term=Terminal, MM=Multimodal, Oth=Others.

Method	Overall	Cost ($)	Hit (M)	Miss (M)	Output (M)
Vanilla	79.2	7.24	25.015	5.943	0.202
+ Global Level	79.9	4.22	26.716	1.589	0.227
+ Local Level	81.3	2.79	8.551	1.549	0.219

Table 3: Progressive ablation of TokenPilot components on `PinchBench` in continuous mode. "Hit" and "Miss" denote the token counts for cache hits and cache misses.

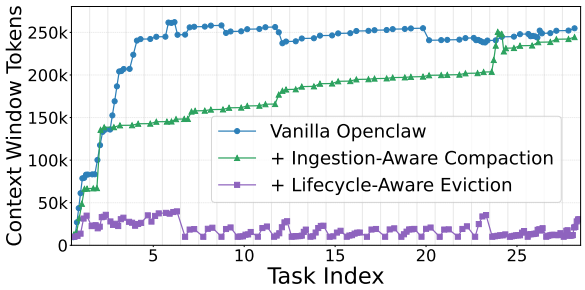


Figure 3: Per-call context token volume across a continuous `Meeting Analysis` session.

on `Claw-Eval` while maintaining competitive task accuracy. Text-level compression methods lower expenditures but consistently degrade task performance due to aggressive semantic pruning. Conversely, while dynamic frameworks preserve execution quality, they fail to regulate prompt cache and incur cache miss penalties. TokenPilot successfully bypasses these limitations, achieving optimal economy without sacrificing task effectiveness.

Continuous Mode. Under continuous task streams, long-horizon text accumulation severely amplifies these macro performance gaps. On `PinchBench`, TokenPilot sustains a top performance score of **81.3** at a minimal expenditure of **$2.79**, restricting cache misses to **1.549M** tokens. On `Claw-Eval`, unrestricted history growth causes catastrophic cost inflation, forcing Vanilla expenditures to rocket to **$81.52**. TokenPilot slashes this operational cost down to **$10.58**, demonstrating robust scalability and systemic superiority over existing paradigms in deployment environments.

4.3 Ablation Study

Table 3 and Figure 3 present the progressive contribution of each TokenPilot component against the Vanilla baseline, which lacks proactive entry-gate regulation. As shown in Figure 3, the baseline context size climbs rapidly and remains persistently high across the execution horizon, despite its built-in compaction constraining the peak volume.

Integrating **Ingestion-Aware Compaction** via rule-based pruning and prefix stabilization slashes total expenditure from **$7.24** to **$4.22**. This enhancement is validated by a sharp reduction in cache miss tokens from **5.943M** to **1.589M**. Explic-

Method	Overall	Cost ($)	Hit (M)	Miss (M)	Output (M)
Vanilla	80.47	8.31	6.184	8.753	0.285
+ Cache Stabilization	80.81	4.35	12.948	2.818	0.295
+ Reduction Pass	**80.92**	**2.87**	8.700	1.493	0.245
- Recovery Tool	77.12	4.03	11.780	2.539	0.276

Table 4: Component-level analysis of *Ingestion-Aware Compaction* on PinchBench in isolated mode.

Cache Hit	PinchBench		Claw-Eval	
	Vanilla	w/ Stable	Vanilla	w/ Stable
0	8.1%	2.44%	100.0%	3.7%
2,048	91.9%	0.0%	0.0%	0.0%
5,120	0.0%	77.24%	0.0%	0.0%
5,888	0.0%	0.0%	0.0%	0.6%
6,144	0.0%	0.0%	0.0%	59.6%
6,656	0.0%	0.0%	0.0%	31.1%
7,168	0.0%	0.0%	0.0%	5.0%
12,288	0.0%	6.50%	0.0%	0.0%
12,800	0.0%	13.82%	0.0%	0.0%

Table 5: Distribution of warm-start cache hit tokens at the first inference call of each task across benchmarks.

itly, rule-based pruning dampens context peaks by filtering verbose environmental noise before insertion, while stabilization converts expensive pre-fills into cache hits by enforcing reliable layout reuse across consecutive tasks.

Layering *Lifecycle-Aware Eviction* further minimizes expenditures to **$2.79** while maintaining the overall score. This component triggers a **65.0%** reduction in cache hit tokens from **26.716M** to **8.551M**, demonstrating that TokenPilot tightly caps the active memory footprint. The periodic tracking drops in Figure 3 confirm that our conservative batch-turn schedule executes precise memory offloading only when task residual utility expires.

4.4 Analysis of Ingestion-Aware Compaction

We isolate the individual impacts of prefix stabilization and context reduction by benchmarking tasks independently in Table 4. Merely introducing stable placeholders cuts the baseline cost from **$8.31** to **$4.35** by converting cache misses into cache hits, while layering reduction passes further minimizes the expenditure to **$2.87**.

Prefix Stabilization Facilitates Warm Starts. Physical cache continuity is disrupted by universal volatile fields, including directory paths and timestamps, and environment-specific tool definitions. Universal markers dominate prefix instability on PinchBench, whereas Claw-Eval configurations introduce severe tool-schema jitter. Replacing these dynamic fields with static placeholders transforms cold initializations into immediate warm starts, ensuring successive tasks inherit the accumulated prompt cache.

Method	Overall	Cost ($)	Hit (M)	Miss (M)	Output (M)
Vanilla	79.2	7.24	25.02	5.94	0.20
+ Prefix Stabilization	79.4	6.39	28.02	4.40	0.22
Summary	78.4	7.12	20.69	6.25	0.20
+ Prefix Stabilization	78.9	6.06	24.16	4.23	0.24
Keep-Last-N	79.1	5.66	18.12	4.48	0.21
+ Prefix Stabilization	79.0	5.20	20.47	3.63	0.21

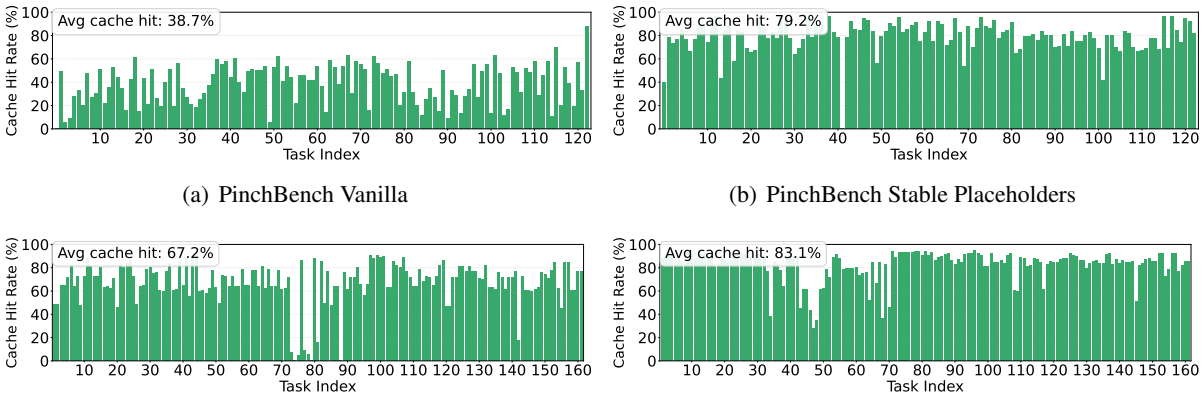
Table 6: Prefix stabilization applied to other baselines on PinchBench in continuous mode.

As validated by Table 5, stable placeholders migrate the vast majority of tasks from minimal baseline token allocations to high-capacity warm starts across both platforms. Consequently, Figure 4 shows that the macro cache hit rate rises from **38.7%** to **79.2%** on PinchBench, and from **67.2%** to **83.1%** on Claw-Eval.

Table 6 further applies stabilization to three representative context-management strategies under the same category-grouped continuous protocol. It reduces cost for Vanilla, Summary, and Keep-Last-N. The smaller gain for Keep-Last-N reflects a remaining limitation of fixed-window truncation: stabilization preserves the invariant core prefix, but cannot recover task-specific cache reuse after the retained history window shifts.

Context Reduction Compounds Savings via Fallback Loops. Context reduction exploits a compounding logic: tokens eliminated at the framework boundary never accumulate in multi-turn windows. Figure 5 demonstrates that our dual reduction passes directly strip heavy payloads across heterogeneous tasks, removing up to **115k** characters of structural noise in oss_alternative_research via HTML slimming, and up to **883k** characters of terminal logs in meeting_gov_recommendations via execution truncation. This targeted compaction effectively trims the sequence footprint while sustaining task accuracy at **80.9**.

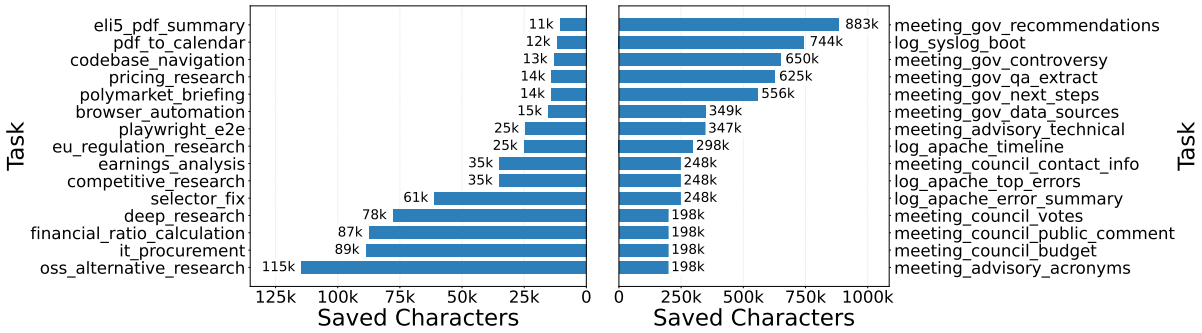
The recovery tool is essential to sustaining this performance boundary. Completely disabling it triggers a capability drop from **80.9** to **77.1** while inflating expenditures to **$4.03**. Without full content access, the agent executes compensatory retries that append fresh tool feedback and overwhelm rule-based compaction. The recovery mechanism breaks this inflationary cycle by providing on-demand payload access, preserving task effectiveness while halting uncontrolled context growth.



(a) PinchBench Vanilla (b) PinchBench Stable Placeholders

(c) Claw-Eval Vanilla (d) Claw-Eval Stable Placeholders

Figure 4: Per-Task Cache Hit Rate on PinchBench and Claw-Eval.



(a) HTML Slimming Pass Reduction (b) Exec Output Truncation Pass Reduction

Figure 5: Per-Task Character Savings from Reduction Passes.

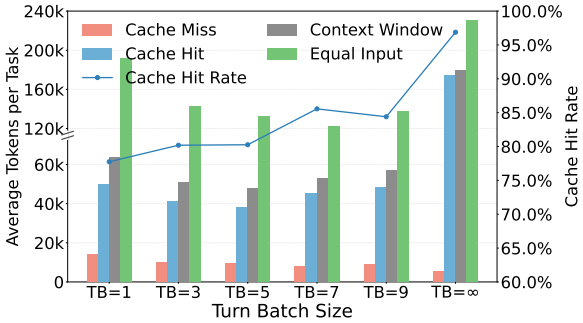


Figure 6: Average per-task Cache Miss, Cache Hit, Context Window, and Equal Input tokens, alongside Cache Hit Rate, across turn batch sizes $B \in \{1, 3, 5, 7, 9, \infty\}$ on the `Meeting Analysis` category of `PinchBench` in continuous mode.

4.5 Analysis of Lifecycle-Aware Eviction

Batch Trigger Intervals Regulate Context Size and Cache Stability. We evaluate eviction trigger frequencies in Figure 6, where *Context Window* tracks physical text accumulation, and *Equal Input* measures the equivalent monetary cost by discounting cache hits relative to expensive pre-fill misses. The *Cache Hit Rate* line reflects the percentage of cached tokens relative to total ingestion.

Completely disabling eviction ($B = \infty$) causes both metrics to peak, confirming that unbounded history growth escalates deployment expenditures. While lifecycle eviction lowers both curves, a hyperactive schedule ($B = 1$) triggers premature truncation that disrupts layout consistency and inflates cache misses. Conversely, larger batch sizes preserve prefix continuity to improve cache hit rates. Balancing task accuracy, memory reduction, and API call times, $B = 3$ constitutes the empirical optimum by preventing memory inflation while securing reliable hardware-level cache reuse.

Retention Signal Comparison. To isolate the retention decision from the eviction implementation, Table 7 compares our model-based residual-utility estimator with four interpretable signals. Recent-turn retains tasks active within the last three turns; recent-file-access retains tasks associated with a file read within the last three turns and not subse-

Retention Signal	Overall	Cost ($)	Hit (M)	Miss (M)	Output (M)
Category-grouped streams					
Recent-turn	78.97	5.86	12.94	5.00	0.25
Recent-file-access	74.11	5.17	14.49	3.98	0.24
Tool-dependency	78.06	6.82	20.85	5.63	0.23
Task-category	78.71	5.31	19.71	3.66	0.24
Residual-utility	78.81	4.69	17.16	2.99	0.26
Mixed-category streams					
Recent-turn	69.80	5.79	18.34	4.31	0.26
Recent-file-access	67.60	6.38	23.95	4.38	0.29
Tool-dependency	67.71	7.04	25.54	5.14	0.28
Task-category	67.59	6.77	23.72	5.03	0.27
Residual-utility	69.16	5.25	14.83	3.85	0.28

Table 7: Controlled comparison of lifecycle retention signals on PinchBench in category-grouped and mixed-category continuous streams. Observation reduction is disabled, and all methods share prefix stabilization and the same block-level eviction executor.

quently overwritten; tool-dependency retains tasks sharing a tool family with the last three turns; and the task-category retains the current contiguous category segment. All variants use the same runtime, backbone, continuous protocol, prefix stabilization, and block-level eviction executor, with observation reduction disabled.

Because grouping same-category tasks into contiguous sessions can favor cross-task residual utility and cache reuse, we additionally evaluate heterogeneous streams constructed by shuffling tasks across categories (see Appendix A.5 for detailed setting). On the category-grouped streams, the residual-utility estimator achieves the lowest cost of $4.69 while retaining a score comparable to the strongest heuristic. On mixed-category streams, it remains the lowest-cost signal and reduces cost to $5.25 relative to Recent-turn. The narrower margin is expected because frequent task and tool-schema changes leave less residual utility to exploit. Since all variants share the same execution path, these differences arise from the retention signal rather than from different deletion or rewrite mechanisms.

Residual Utility Prevents Over-Eviction. We complement this aggregate comparison with a variant that disables residual utility estimation and immediately purges historical segments upon sub-task completion. Figure 7 traces a representative four-task session in which successive tasks manipulate a shared file named `transcript.md`.

Under TokenPilot, the primary task ingests `transcript.md` and anchors its structure in memory. When downstream tasks arrive, the framework estimator infers from tool dependency patterns that this historical segment retains residual utility, preserving its slots despite sub-task completion. Consequently, tools directly locate target content blocks including personnel sections and roadmap data without re-exploring background knowledge.

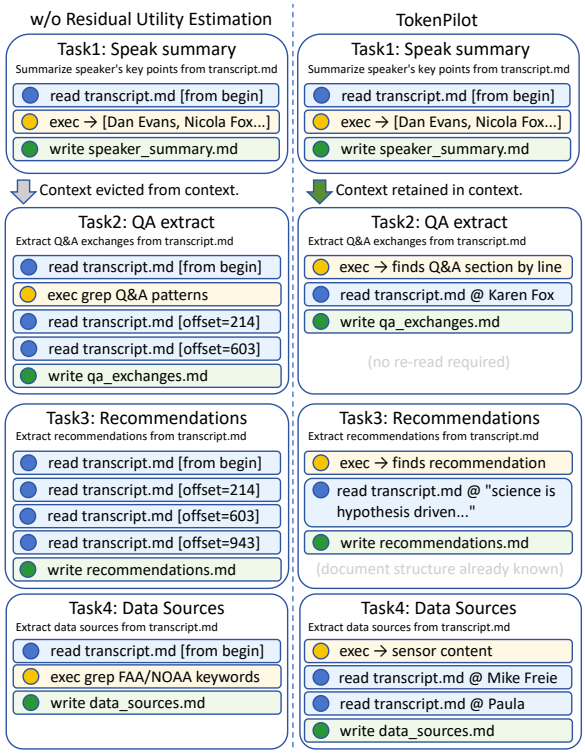


Figure 7: Tool call patterns across a four-task session on `transcript.md` under TokenPilot and a variant without residual utility estimation. TokenPilot retains the context segment after task completion, enabling subsequent tasks to directly access relevant document sections. Without residual utility estimation, each task re-reads files from the beginning and issues multiple sequential partial reads to locate the same content.

Conversely, the variant without residual utility estimation triggers eviction immediately upon local execution completion. Each subsequent task thus inherits a cold window, forcing it to rediscover the document architecture via redundant full file reads and sequential scans. This contrast demonstrates that the residual utility buffer acts as a valve to preserve document knowledge across distinct tasks, enabling targeted access while blocking the overhead of continuous context re-exploration.

5 Related Work

Static Content Compression and Abstraction. From the perspective of maximizing utility density within the context window, a prominent strategy focuses on filtering or restructuring historical trajectories at the ingestion stage to preserve the

premium context budget. To eliminate linguistic redundancy and limit token footprints, early efforts successfully prune non-essential text units or compress structural prompt representations (Jiang et al., 2023; Pan et al., 2024; Li et al., 2023; Jia et al., 2026; Kong et al., 2025). Extending this content-reduction philosophy to a macro scale, passive memory retrieval systems manage the runtime working memory by offloading entire session histories to external databases, selectively recalling high-utility historical fragments while filtering out the remaining interactions (Packer et al., 2023; Chhikara et al., 2025; Zhong et al., 2024; Fang et al., 2025). Rather than applying hard textual pruning, complementary approaches enhance information density by converting raw trajectories into structured, high-level semantic abstractions. To maintain holistic task coherence, these frameworks effectively substitute continuous interaction footprints with localized recursive summaries, hierarchical sub-goal graphs, or episodic, temporal knowledge graphs to secure macro-level guidance across elongated session horizons (Ehrlich and Blackman, 2026; Wu et al., 2025; Hu et al., 2025a; Liu et al., 2025b; Li et al., 2026; Rasmussen et al., 2025).

Dynamic and Runtime Scheduling. Another significant line of research treats the context window as a fluid operating system resource, managing context segments in real time to align with the agent's immediate execution states. To optimize token distribution during live sessions, modern frameworks successfully execute runtime demand paging, adaptive parallel routing, and context isolation based on real-time trajectories and budget constraints (Mason, 2026; Feng et al., 2026; Qian et al., 2026; Wu et al., 2026; Ye et al., 2025b; Sun et al., 2025; Xu et al., 2026a). For long-horizon planning, advanced architectures manage memory as a self-organizing operating system or introduce virtual memory abstractions to dynamically secure stateful data residency and tool durability (Xu et al., 2026b; Deng et al., 2026; Chen et al., 2026; Fang et al., 2026; Li et al., 2025b; Kang et al., 2025; Rafique and Bindschaedler, 2026). Recently, this runtime scheduling paradigm has expanded to multi-agent environments, utilizing decentralized role-aware routing, centralized experience caching, or cross-context KV-cache communication topographies to minimize distributed token footprints across collaborative swarms (Liu et al., 2025a; Zhang et al., 2026; Han et al., 2025; Ye et al., 2025a).

6 Conclusion

We presented **TokenPilot**, a dual-granularity framework that reconciles text reduction with strict prompt cache alignment. By separating memory management into global ingestion-aware compaction and local lifecycle-aware eviction, our framework successfully stabilizes dynamic layouts while conservatively offloading context based on task-level residual utility. Empirical evaluations on both `PinchBench` and `Claw-Eval` demonstrate that **TokenPilot** drastically cuts inference expenditures under both isolated and continuous operational modes without sacrificing task effectiveness, offering a scalable and highly cost-efficient foundation for long-horizon agent systems.

Limitations

Despite its strong performance, TokenPilot has several limitations. The model-based estimator may misclassify context segments under highly ambiguous or sparse interaction patterns, and the frequency threshold τ and batch size B may require tuning for different deployment environments and task distributions. The prefix stabilization component additionally relies on backend support for prefix caching, providing no benefit to providers without this feature. Finally, our continuous mode evaluation groups same-category tasks into contiguous sessions to reflect domain-specific workflow environments; heavily shuffled or highly heterogeneous mixed-category task streams may naturally exhibit lower prefix reuse rates due to persistent tool schema mutations, which we leave as an important direction for future investigation.

Acknowledgements

We would like to express our sincere gratitude to the anonymous reviewers for their thoughtful and constructive feedback. This work was supported by the New Generation Artificial Intelligence-National Science and Technology Major Project (2025ZD0122802), National Natural Science Foundation of China (No. 62576307, No. NSFCU23B2055, No. NSFCU19B2027), the Fundamental Research Funds for the Central Universities (226-2023-00138), the Yongjiang Talent Introduction Programme (2021A-156-G), and the Information Technology Center and State Key Lab of CAD&CG, Zhejiang University.

References

Anthropic. 2025. Claude code.

Yijun Chen, Boyi Xiao, Yixian Zhao, Haoting Xia, Buqiang Xu, Jizhan Fang, Yanya Li, Yaqi Zheng, Xuehai Wang, Zirui Xue, Liuxin Zhang, Hui Li, and Ningyu Zhang. 2026. Lightmem-ego: Your AI memory for everyday life. *CoRR*, abs/2607.11487.

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

Xinle Deng, Yida Xue, Xiangyuan Ru, Yijun Chen, Buqiang Xu, Mingjun Mao, Xinjie Liu, Haoming Xu, Shuofei Qiao, Mengru Wang, Chen Jiang, Yuchen Eleanor Jiang, Lizhong Wang, Jason Wang, Li Zeng, Haofen Wang, Guilin Qi, Huajun Chen, and Ningyu Zhang. 2026. Mobilemem: Learning from a year of mobile experiences. *Preprint*, arXiv:2608.13606.

Clint Ehrlich and Theodore Blackman. 2026. LCM: Lossless context management. Technical report, Voltropy PBC.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. 2025. Lightmem: Lightweight and efficient memory-augmented generation. *CoRR*, abs/2510.18866.

Jizhan Fang, Buqiang Xu, Zhixian Wang, Haoliang Cao, Xinle Deng, Baohua Dong, Hangcheng Zhu, Ruohui Huang, Gang Yu, Ying Wei, Guozhou Zheng, Feiyu Xiong, Haofen Wang, Huajun Chen, and Ningyu Zhang. 2026. Rethinking memory as continuously evolving connectivity. *CoRR*, abs/2605.28773.

Zhaopeng Feng, Liangcai Su, Zhen Zhang, Xinyu Wang, Xiaotian Zhang, Xiaobin Wang, Runnan Fang, Qi Zhang, Baixuan Li, Shihao Cai, Rui Ye, Hui Chen, Yong Jiang, Joey Tianyi Zhou, Chenxiong Qian, Pengjun Xie, Bryan Hooi, Zuozhu Liu, and Jingren Zhou. 2026. Agentswing: Adaptive parallel context management routing for long-horizon web agents. *CoRR*, abs/2603.27490.

Dongge Han, Camille Couturier, Daniel Madrigal Díaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. 2025. Legomem: Modular procedural memory for multi-agent LLM systems for workflow automation. *CoRR*, abs/2510.04851.

Chuanrui Hu, Xingze Gao, Zuyi Zhou, Dannong Xu, Yi Bai, Xintong Li, Hui Zhang, Tong Li, Chong Zhang, Lidong Bing, and Yafeng Deng. 2026. Evermemos: A self-organizing memory operating system for structured long-horizon reasoning. *CoRR*, abs/2601.02163.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2025a. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 32779–32798. Association for Computational Linguistics.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, and 28 others. 2025b. Memory in the age of AI agents. *CoRR*, abs/2512.13564.

Haoxiang Jia, Earl T. Barr, and Sergey Mechtaev. 2026. Compressing code context for llm-based issue resolution. *CoRR*, abs/2603.28119.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 13358–13376. Association for Computational Linguistics.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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

Kilo AI Team. 2026. Pinchbench: Real-world benchmarks for ai coding agents.

Zhenglun Kong, Yize Li, Fanhu Zeng, Lei Xin, Shvat Messica, Xue Lin, Pu Zhao, Manolis Kellis, Hao Tang, and Marinka Zitnik. 2025. Token reduction should go beyond efficiency in generative models - from vision, language to multimodality. *CoRR*, abs/2505.18227.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, and 2 others. 2025a. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *CoRR*, abs/2510.25726.

Xiaoxi Li, Wenxiang Jiao, Jiarui Jin, Guanting Dong, Jiajie Jin, Yinuo Wang, Hao Wang, Yutao Zhu, Ji-Rong Wen, Yuan Lu, and Zhicheng Dou. 2026. Deepagent: A general reasoning agent with scalable toolsets. In *Proceedings of the ACM Web Conference 2026, WWW 2026, Dubai, United Arab Emirates, originally scheduled for April 13-17, 2026, rescheduled for June 29 - July 3, 2026*, pages 2219–2230. ACM.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. 2023. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 6342–6353. Association for Computational Linguistics.

Zhiyu Li, Shichao Song, Chenyang Xi, Hanyu Wang, Chen Tang, Simin Niu, Ding Chen, Jiawei Yang, Chunyu Li, Qingchen Yu, Jihao Zhao, Yezhaohui Wang, Peng Liu, Zehao Lin, Pengyuan Wang, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, and 20 others. 2025b. Memos: A memory OS for AI system. *CoRR*, abs/2507.03724.

Jun Liu, Zhenglun Kong, Changdi Yang, Fan Yang, Tianqi Li, Peiyan Dong, Joannah Nanjekye, Hao Tang, Geng Yuan, Wei Niu, Wenbin Zhang, Pu Zhao, Xue Lin, Dong Huang, and Yanzhi Wang. 2025a. Rcr-router: Efficient role-aware context routing for multi-agent LLM systems with structured memory. *CoRR*, abs/2508.04903.

Shukai Liu, Jian Yang, Bo Jiang, Yizhi Li, Jinyang Guo, Xianglong Liu, and Bryan Dai. 2025b. Context as a tool: Context management for long-horizon swe-agents. *CoRR*, abs/2512.22087.

Tony Mason. 2026. The missing memory hierarchy: Demand paging for LLM context windows. *CoRR*, abs/2603.09023.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. 2025. A survey of context engineering for large language models. *CoRR*, abs/2507.13334.

Mike A. Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, Estefany Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis-Mastromichalakis, and 66 others. 2026. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *CoRR*, abs/2601.11868.

OpenAI. 2026. Codex: AI coding partner from OpenAI.

OpenClaw. 2026. Openclaw.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. 2023. Memgpt: Towards llms as operating systems. *CoRR*, abs/2310.08560.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Findings of ACL, pages 963–981. Association for Computational Linguistics.

Hongjin Qian, Zhao Cao, and Zheng Liu. 2026. Memobrain: Executive memory as an agentic brain for reasoning. *CoRR*, abs/2601.08079.

Mofasshara Rafique and Laurent Bindschaedler. 2026. Clawvm: Harness-managed virtual memory for stateful tool-using LLM agents. In *Proceedings of the Sixth European Workshop on Machine Learning and Systems, EuroMLSys 2026, Edinburgh, Scotland, UK, April 27-30, 2026*, pages 1–12. ACM.

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

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. 2025. Scaling long-horizon LLM agent via context-folding. *CoRR*, abs/2510.11967.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. 2025. Resum: Unlocking long-horizon search intelligence via context summarization. *CoRR*, abs/2509.13313.

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. 2026. Contextbudget: Budget-aware context management for long-horizon search agents. *CoRR*, abs/2604.01664.

Binyan Xu, Haitao Li, and Kehuan Zhang. 2026a. LLM agents are latent context managers: Eliciting self-managed context via a proprioceptive dashboard. *CoRR*, abs/2606.30005.

Buqiang Xu, Yijun Chen, Jizhan Fang, Ruobin Zhong, Yunzhi Yao, Yuqi Zhu, Lun Du, and Shumin Deng. 2026b. Structmem: Structured memory for long-horizon behavior in llms. *CoRR*, abs/2604.21748.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. 2026. Claw-eval: Toward trustworthy evaluation of autonomous agents. *CoRR*, abs/2604.06132.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. 2025a. KVCOMM: online cross-context kv-cache communication for efficient llm-based multi-agent systems. *CoRR*, abs/2510.12872.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Siheng Chen, Jingren Zhou, and Yong Jiang. 2025b. Agentfold: Long-horizon web agents with proactive context management. *CoRR*, abs/2510.24699.

Ruizhe Zhang, Xinke Jiang, Zhibang Yang, Zhixin Zhang, Jiaran Gao, Yuzhen Xiao, Hongbin Lai, Xu Chu, Junfeng Zhao, and Yasha Wang. 2026. Stackplanner: A centralized hierarchical multi-agent system with task-experience memory management. *CoRR*, abs/2601.05890.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. 2024. Sglang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 19724–19731. AAAI Press.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

PinchBench		Claw-Eval (General)	
Category	**#**	**Category**	**#**
Productivity	8	Workflow	47
Research	12	Ops	31
Writing	6	Finance	14
Coding	14	Office QA	10
Analysis	12	Communication	8
CSV Analysis	26	Productivity	7
Log Analysis	6	Operations	6
Meeting Analysis	28	Safety	5
Memory	2	Terminal	5
Skills	6	Multimodal	4
Integrations	3	Others	24
Total	**123**	**Total**	**161**

Table 8: Statistics for `PinchBench` and `Claw-Eval`.

A Appendix

A.1 Dataset Configurations

To evaluate `TokenPilot`, we utilize two realistic agent benchmarks: `PinchBench` and `Claw-Eval`.

`PinchBench` is a real-world evaluation suite comprising 11 distinct task categories and 123 tasks in total. To account for the continuous rolling updates in the upstream repository, we benchmark our framework on a frozen snapshot of the benchmark.[2]

`Claw-Eval` is a containerized agent evaluation platform executed within isolated sandbox environments. We evaluate on its **General** task group, which encompasses 161 multi-step service orchestration and standalone analytical tasks.

For both benchmarks, we group same-category tasks into contiguous, uninterrupted single sessions to faithfully simulate realistic continuous multi-task agent execution trajectories. The detailed structural statistics of these evaluation platforms are compiled in Table 8.

A.2 Evaluation Metrics and Cost Modeling

Task Score Execution Framework. We strictly adhere to the native evaluation frameworks provided by each respective benchmark to compute task performance.

For `Claw-Eval`, the scoring pipeline executes an integrated multi-dimensional protocol evaluating *Completion* (s_{comp}), *Safety* (s_{safe}), and *Robustness* (s_{rob}) as coupled parameters grounded in multi-channel auditable trajectory evidence, including

[2] https://github.com/pinchbench/skill/commit/0347a7f1736a9c33b5fe831e27d1d6ee9b576221

service audit logs, environment snapshots, and execution traces. Formally, the final task score is mathematically formulated as follows:

$$\text{Score} = s_{\text{safe}} \times (0.80 \cdot s_{\text{comp}} + 0.20 \cdot s_{\text{rob}}) \quad (10)$$

where Safety acts as a strict multiplicative gate, while Completion and Robustness represent the primary goal-directed execution quality and secondary error-recovery capability under controlled service perturbations respectively. This fine-grained rubric triangulation yields continuous partial credits rather than trivial binary verdicts.

For `PinchBench`, the framework aggregates task-specific verification checks on output deliverables to assess the agent's goal-directed capability.

Crucially, when evaluating system trajectories in **Continuous Mode**, we implement a trajectory slicing mechanism that automatically partitions the continuous session transcript file into task-specific segments based on original task boundaries. Each sliced segment is then fed independently into the corresponding benchmark grader, ensuring that the evaluation logic for continuous task streams remains strictly identical and mathematically comparable to that of the **Isolated Mode**.

Inference Cost Modeling. To calculate the runtime inference cost across sequential execution sessions, we implement a monetary cost metric based on commercial deployment pricing. The total inference cost is calculated as follows:

$$\text{Cost} = |\mathcal{C}'_{\text{hit}}| \cdot p_{\text{hit}} + |\mathcal{C}'_{\text{miss}}| \cdot p_{\text{miss}} + \mathcal{H}_{\text{out}} \cdot p_{\text{out}} \quad (11)$$

where $|\mathcal{C}'_{\text{hit}}|$ and $|\mathcal{C}'_{\text{miss}}|$ represent the number of input tokens that hit or miss the KV cache backend respectively, and $\mathcal{H}_{\text{out}}$ represents the length of the generated agent responses. Following the official pricing tiers of `GPT-5.4-mini`, the price parameters are set to $p_{\text{hit}} = \$0.075/\text{M}$ tokens for cache hits, $p_{\text{miss}} = \$0.75/\text{M}$ tokens for cache misses, and $p_{\text{out}} = \$4.50/\text{M}$ tokens for outputs.

A.3 Baseline Configurations

To ensure reproducibility, we document the configurations for all evaluated baselines.

① **Vanilla** runs on OpenClaw without any extra context management, with a maximum context window of 500k tokens and a compaction trigger ratio of 0.5.

② **LLMLingua-2** applies token-level compression using a small language model, with a compression ratio of 0.6.

③ **SelectiveContext** applies sentence-level compression based on self-information, with a compression ratio of 0.4.

④ **LCM** applies lossless compaction via hierarchical summarization, triggered when context reaches 75% of the context window. Each leaf chunk accumulates up to 80k tokens before summarization, retaining the 64 most recent turns in full fidelity.

⑤ **Pichay** uses utility-driven demand paging with the following thresholds: advisory zone at 60k tokens, involuntary eviction zone at 100k tokens, and a hard cap at 120k tokens. Tool results older than 4 user turns are eligible for compression, with a minimum eviction size of 500 bytes.

⑥ **Summary** compresses interaction history into hierarchical summaries when context reaches 40% of the 500k token window.

⑦ **MemoBrain** maintains a memory budget of 100k tokens and triggers recall when estimated context length reaches 35% of the memory budget.

⑧ **AgentSwing** selects among three candidate strategies (discard-all, keep-last-n, summary) via lookahead simulation of 3 future turns. It triggers at a token ratio of 0.2 within a 200k context window, retaining the 5 most recent turns under the keep-last-n strategy.

⑨ **Keep-Last-N** retains the most recent $N = 5$ turns when context reaches 40% of the 500k token window.

⑩ **MemOS** limits retrieval to 20 items per recall turn to control token costs. It filters candidate memories by exposing at most 500 characters per item to the validation model, while restricting individual memory ingestion to a maximum of 20,000 characters per message.

A.4 Implementation Details

This section documents the underlying engineering configurations, threshold parameterizations, and prompting architectures of `TokenPilot` to facilitate exact reproducibility. We detail the deterministic mechanics for cache stabilization and observation reduction in alignment with our system design, followed by their fine-grained numerical hyperparameters and specific base model assignments. Finally, we present the complete system prompt templates utilized for our state estimation pipeline.

Cache Stabilization. To ensure the prompt prefix remains byte-identical across consecutive turns, `TokenPilot` standardizes and restructures the input

layout before each inference call.

First, runtime-volatile text fields within the system prompt messages, such as working directory paths, active timestamps, and transient session identifiers, are substituted with static, stable placeholders. Second, since distinct tasks often require different tool configurations, leaving tool definitions inside the primary system prompt introduces structural variations that break baseline prefix matching. To mitigate this positioning jitter, we systematically relocate the tool definitions and schemas downstream, placing them at the end of the system prompt message directly alongside the dynamic context block containing the original values of the volatile fields.

Observation Reduction. To suppress textual redundancy and regulate the per-turn input volume, we implement a sequence of rule-based reduction passes targeting low-utility tool result messages before they enter the canonical history. Specifically, repeated tool call results are deduplicated via hashing, while oversized tool call parameters and long execution outputs are truncated beyond a fixed token threshold. To prevent critical information loss from hard truncation, we equip the agent with a dedicated recovery tool, allowing it to dynamically retrieve full execution payloads when necessary. For multimodal and web-browsing interactions, web-fetched content undergoes HTML slimming to remove non-essential markup and attributes, and embedded images are downsampled to minimize their respective token footprint. Finally, a general formatting pass cleans up the remaining layout variations by removing code fences, invalid format symbols, and line number prefixes from code outputs, alongside normalizing continuous whitespace characters.

Hyperparameters for Context Reduction. For the rule-based context reduction passes, the specific numerical thresholds and fine-grained hyperparameter configurations are parameterized as follows:

① **Activation Gates:** The minimum character count to trigger before-call reduction is set to `triggerMinChars = 2200`, and candidate tool-like fragments are routed to the module only when exceeding `maxToolChars = 1200`.

② **Execution Output Truncation:** The global truncation threshold for generic tool feedback is bounded at `50k` characters. For tool-specific profiles, the limits are set to `30k` for `bash/shell/powershell`, `20k` for `grep/rg`, `10k` for `mcp_auth`, and `100k` for `glob/write/edit`, while `read/file_read` is permitted an unconstrained capacity (`Infinity`). Truncated outputs consistently retain an initial `600`-character prefix and a terminal `400`-character suffix as a preview block, which can be fully recalled via the recovery tool when triggered by the agent.

③ **Deduplication and Frequency Limits:** For the `repeated_read_dedup` pass, redundant read operations are substituted with the same `600/400` preview block. To suppress infinite loop behaviors and redundant footprint accumulation, the maximum sequential execution frequency for any identical tool call within a rolling tracking window is strictly capped at `5`. Multimodal constraints under `image_downsample` restrict standard bitmap layouts to a maximum size of `100KB` and vector-based `SVG` documents to `50KB`.

④ **Layout Cleaning Constraints:** File path markers are clipped via `path_truncation` to a maximum length constraint of `80` characters. The remaining syntactic layout transformations, including `html_slimming`, `format_slimming`, `format_cleaning`, and `line_number_strip`, are deterministically executed without extra numerical hyperparameters.

Model and Hyperparameter Configurations. To ensure a rigorous and fair empirical comparison, all baseline agent architectures and our primary inference execution module are deployed under identical base model configurations, specifically utilizing `GPT-5.4-mini`. For the internal state estimation and context utility metrics, we employ `Qwen3.5-35B-A3B` as the dedicated estimator model. The batch-turn tracking window for interval context processing is set to 3.

Prompt Templates for the State Estimator To provide full architectural transparency and ensure exact reproducibility, we detail the core system prompt configurations injected into our `Qwen3.5-35B-A3B` state estimator. The estimator operates as a structured semantic tracking pipeline that processes continuous session trajectories and outputs incremental semantic deltas in a validation-ready JSON format. Depending on whether the intermediate residual utility gating layer is active, the system configures the estimator prompt under two distinct tracking paradigms:

As illustrated in Figure 8, the full operational configuration of `TokenPilot` deploys a joint classification-and-eviction scheme. The system

Prompts for State Estimator with Residual Utility Gating

```
[USER]: You are a task-state estimator for a long-running agent session. Your job is to update global task
state incrementally. You must only return a JSON object. Do not output a full registry; you must return only
a semantic delta, not a registry patch. The input is incremental, but the task registry is global.
Each update may modify the lifecycle of any existing task in the session, including older tasks that are not
directly covered by the newest delta. You must backfill task ownership for every covered turn in the delta
window. Never invent turn ids that are not present in the provided delta.
When the newest covered turn contains a new top-level user request, you must decide whether it starts a new
task. If the newest user request is materially different from the objective of the current active task,
create a new task update anchored to the first covered turn of that new request instead of extending the old
task. If a newer top-level user request starts a different task, do not keep an older unrelated one-shot task
active.
When an older task already has delivery evidence, no unresolved questions, and is not covered by the current
delta, you may mark it evictable instead of leaving it active. When the hints include
evictableCandidateTaskIds and the newest covered turn clearly starts or finishes a different task, you should
usually emit lifecycle-only updates that mark those candidate tasks evictable in the same response.
Do not wait for another future turn to mark an obviously finished older task evictable once a newer distinct
task has already taken over the session. Never mark a task evictable unless it is already completed or you
are simultaneously providing clear completionEvidence. Never use evictable for a task that still lacks
completion evidence, still has unresolved questions, or is obviously in progress. Use completed only when the
task is finished but still likely to be referenced again immediately. Use evictable only when the task is
finished, has completionEvidence, has no unresolved questions, and the session has already moved on to a
different task.
The delta may include completedTaskSummaries when older completed tasks have been compressed out of the
active estimator context. Treat completedTaskSummaries as stable background memory and prefer keeping the
currently unresolved task as one continuous task unless the newest user request clearly starts a new
objective.
Output schema must be exactly:
{"baseVersion": number, "taskUpdates": SemanticTaskUpdate[]}
SemanticTaskUpdate must use exactly these fields:
{"taskId": string, "title"?: string, "objective": string, "lifecycle":
"active"|"blocked"|"completed"|"evictable", "coveredTurnAbsIds"?: string[], "completionEvidence"?: string[],
"unresolvedQuestions"?: string[], "currentSubgoal"?: string, "evictableReason"?: string}
coveredTurnAbsIds is required when creating a new task or extending task ownership to new turns.
coveredTurnAbsIds may be omitted or empty for lifecycle-only updates on existing tasks. If lifecycle is
completed or evictable, include completionEvidence unless the existing registry entry already has strong
completion evidence. If lifecycle is evictable, include evictableReason as one short sentence.
Do not output registry patch fields such as upsertTasks, activeTaskIds, completedTaskIds, evictableTaskIds,
upsertTurnToTaskIds, transitions, span, or lastProcessedTurnSeq. Do not use alternate field names such as
status, description, action, fromTurnSeq, toTurnSeq, task_created, or task_progressed.
```

Figure 8: System prompt template for TokenPilot's Primary Estimator, featuring joint tracking of completion evidence and explicit cache eviction signaling.

prompt instructs the estimator to evaluate ongoing tool dependencies and cross-turn data reuse patterns before rendering an expiration judgment. Crucially, it introduces a three-state transition matrix by enforcing an explicit `evictable` lifecycle token alongside standard `active` and `completed` identifiers. By verifying delivery evidence and historical dependencies across the session trajectory, this prompt establishes a text-level buffer gate. Historical context segments are only flagged for cache clearance when the estimator explicitly infers that their operational task relevance has fully expired.

To systematically isolate the impact of our gating mechanism, the ablated configuration documented in Figure 9 strips out the cache-aware buffering layer. Under this setup, the prompt restricts the model's objective to binary task progression classification. It limits the lifecycle field to `active` or `completed` tokens and prohibits the output of the `evictable` status identifier. Tasks transition directly to `completed` when local delivery evidence is observed, which triggers immediate context purging at the hardware backend. This configuration serves as the baseline to evaluate the precise cost and efficiency gains brought by the residual utility inference mechanism discussed in Section 4.5.

A.5 Mixed-Category Continuous Evaluation

The continuous evaluation in the main experiments groups tasks from the same category into contiguous sessions, modeling domain-focused workflows. To test a more heterogeneous boundary condition, we additionally construct mixed-category streams by shuffling all 123 `PinchBench` tasks with fixed seed 42 and distributing them into ten continuous sessions of 12 or 13 tasks each. This protocol introduces frequent task-category and tool-schema changes and therefore reduces both cross-task residual utility and opportunities for prefix reuse.

Table 9 reports the full-system comparison. TokenPilot attains a score of 69.10, comparable to Vanilla and Summary, while achieving the lowest cost of $4.83, with a 44.7% below Keep-Last-N. The higher absolute cost than in category-grouped streams is expected: adjacent tasks share fewer artifacts and tools, while schema changes shorten

Prompts for State Estimator with Immediate Completion Eviction

```
[USER]: You are a task-state estimator for a long-running agent session. Your
job is only task progression classification, not cache replacement. Only decide
whether each task is active, blocked, or completed. Do not decide eviction
timing. Never output lifecycle=evictable; eviction will be decided separately by
the system.
You must only return a JSON object. Do not output a full registry; you must
return only a semantic delta, not a registry patch. The input is incremental,
but the task registry is global. Each update may modify the lifecycle of any
existing task in the session, including older tasks that are not directly
covered by the newest delta. You must backfill task ownership for every covered
turn in the delta window.
Prefer one task per distinct user request unless the new request is clearly just
a continuation or clarification of the same objective. For a newly created task,
use a stable taskId derived from the first covered turn, typically
<firstTurnAbsId> or <firstTurnAbsId>-task. Use completed when a task is finished
and has delivery evidence, even if it may later be evicted by a separate policy
layer.
Do not rely on completedTaskSummaries or retained completion evidence from prior
tasks; classify each new task only from the current delta and registry state.
Output schema must be exactly:
{"baseVersion": number, "taskUpdates": SemanticTaskUpdate[]}
SemanticTaskUpdate must use exactly these fields:
{"taskId": string, "title"?: string, "objective": string, "lifecycle":
"active"|"blocked"|"completed", "coveredTurnAbsIds"?: string[],
"completionEvidence"?: string[], "unresolvedQuestions"?: string[],
"currentSubgoal"?: string}
coveredTurnAbsIds is required when creating a new task or extending task
ownership to new turns. coveredTurnAbsIds may be omitted or empty for lifecycle-
only updates on existing tasks. If lifecycle is completed, include
completionEvidence unless the existing registry entry already has strong
completion evidence.
Do not output registry patch fields such as upsertTasks, activeTaskIds,
completedTaskIds, evictableTaskIds, upsertTurnToTaskIds, transitions, span, or
lastProcessedTurnSeq. Do not use alternate field names such as status,
description, action, fromTurnSeq, toTurnSeq, task_created, or task_progressed.
```

Figure 9: System prompt template for the Estimator without Residual Utility Estimation, configured to strip out the caching buffer for the ablation study.

Method	Overall	Cost ($)	Hit (M)	Miss (M)	Output (M)
Vanilla	68.77	21.16	17.45	24.21	0.38
Summary	69.12	13.05	9.26	14.86	0.27
Keep-Last-N	67.32	8.73	6.52	9.30	0.28
LCM	66.07	17.41	32.97	18.28	0.27
SelectiveContext	64.02	14.60	24.95	15.25	0.29
LLMLingua-2	65.13	15.80	24.68	16.82	0.30
TokenPilot	69.10	4.83	14.02	3.49	0.26

Table 9: Performance and resource consumption on mixed-category PinchBench streams.

Method	Overall	Cost ($)	Miss (M)	Read (M)	Output (M)
Vanilla	68.77	21.16	24.21	17.45	0.38
+ Prefix Stabilization	67.83	17.09	19.02	20.33	0.29
Summary	69.12	13.05	14.86	9.26	0.27
+ Prefix Stabilization	68.37	12.39	13.58	11.97	0.29
Keep-Last-N	67.32	8.73	9.30	6.52	0.28
+ Prefix Stabilization	66.91	8.21	8.15	9.34	0.31

Table 10: Prefix-stabilized context-management baselines on mixed-category PinchBench streams.

reusable prefixes. Nevertheless, ingestion-side reduction remains effective independent of task order, and lifecycle-aware eviction bounds accumulated history, allowing TokenPilot to retain its cost advantage under heterogeneous streams.

Table 10 shows that prefix stabilization remains beneficial under heterogeneous ordering, reducing cost from $21.16 to $17.09 for Vanilla, from $13.05 to $12.39 for Summary, and from $8.73 to $8.21 for Keep-Last-N. Its smaller gain for fixed-window management highlights the distinction between preserving the invariant core prefix and controlling the mutable task-history suffix: stabilization addresses the former, whereas reduction and lifecycle-aware eviction regulate the latter.