EVERY TOKEN COUNTS: LEARNING WHAT IS WORTH REMEMBERING FOR RESOURCE-CONSTRAINED LIFELONG AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Lifelong agents continually accumulate and reuse experience to improve after deployment, yet in practice they often operate under tight generation, interaction, and memory budgets. Under such stringent joint constraints, we find that existing lifelong-memory methods can become ineffective, with several sophisticated approaches offering little advantage over simple retrieval or even no-memory agents. This reveals a more fundamental problem: resource scarcity makes each memory decision more consequential. When only a small amount of past experience can be retained or reused and opportunities for reasoning and recovery are limited, retrieving a relevant but ineffective memory can consume scarce context and hinder downstream execution. The key challenge is therefore not simply to retrieve relevant experience, but to determine which experiences are actually worth retaining and reusing. To address this challenge, we propose **TokenWorth**, a lightweight online memory framework that learns query-dependent memory worth from sparse execution feedback and applies it consistently to memory retrieval and bounded memory management. Across three LifelongAgentBench environments and both local and API-based deployments, TokenWorth achieves the highest task success rate while using the fewest output tokens among the evaluated methods. These results suggest that effective lifelong adaptation under resource constraints depends not on remembering more, but on learning what is worth remembering.

1 INTRODUCTION

Lifelong agents continually accumulate, retain, and reuse experience from an evolving task stream, enabling continued adaptation after deployment (Zheng et al., 2026). This capability is particularly valuable for long-running interactive agents that must adapt to changing users, environments, and tasks (Cheng & Chin, 2024; Fang et al., 2025; Luo et al., 2025). In practice, however, lifelong adaptation rarely operates with unlimited resources. Generation is constrained by latency, computation, or inference cost; interaction opportunities are finite; and only a bounded amount of historical experience can be stored or supplied to the agent at each decision step (Gupta, 2025; Kwon et al., 2023; Yuan et al., 2026). This raises a fundamental question: *do existing lifelong-memory mechanisms remain effective when generation, interaction, and memory resources are jointly constrained?*

Our evaluation reveals that they often do not. As shown in Figure 1(a), under stringent joint resource constraints, several sophisticated lifelong-memory approaches provide limited improvement over simple retrieval or even no-memory agents, and can substantially underperform in some environments. These results highlight an important desideratum for lifelong agents: *resource robustness*, the ability to preserve effective adaptation when generation, interaction, and memory resources are severely limited. Importantly, this failure cannot be attributed simply to insufficient memory capacity: under the same resource budget, different memory policies can lead to markedly different downstream task performance.

To study this problem systematically, we formalize *resource-constrained lifelong agents*: lifelong agents that must continually improve from experience while operating under bounded generation, interaction, and memory budgets. As illustrated in Figure 1(b), we jointly constrain the resources available for both task execution and experience reuse. Specifically, the agent operates with a

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.

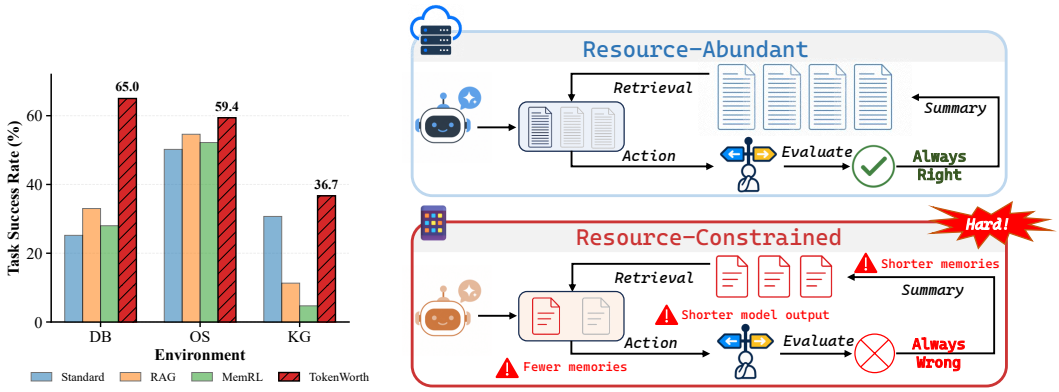


Figure 1: Left: Representative methods under stringent joint resource constraints; TokenWorth remains effective where several memory baselines do not. Right: Resource-constrained lifelong agents operate under bounded generation, interaction, and memory budgets.

bounded output budget and a limited number of interaction rounds, while its external memory is constrained in capacity, the number of retrieved experiences, and the amount of memory that can be injected at each step. These constraints apply to resources available during inference rather than to the capacity of the backbone model itself, allowing us to study lifelong adaptation under a controlled and stringent resource budget.

Why do existing memory mechanisms become fragile under such constraints? When only a small amount of past experience can be retained or supplied to the agent, and opportunities for reasoning and recovery are limited, each memory decision becomes more consequential. In particular, semantic relevance alone may be insufficient for allocating scarce memory resources: an experience can appear relevant to the current query yet provide little downstream utility. Conversely, a useful experience that is not retained or selected cannot be recovered by simply injecting more context or extending the interaction. Resource scarcity therefore turns memory allocation into a more stringent value-estimation problem. The central challenge is no longer merely to retrieve the most relevant past experience, but to determine which experiences are actually *worth* retaining and reusing when memory and execution resources are limited.

To address this challenge, we propose **TokenWorth**, a lightweight online memory framework that learns query-dependent memory worth from sparse execution feedback. TokenWorth combines an interpretable online base scorer with a residual Recursive Feature Machine (RFM), allowing stable global patterns of memory usefulness to be complemented by nonlinear, query-specific corrections. The learned worth is used to guide both memory retrieval and bounded memory-bank management, allocating limited memory resources toward experiences that are most useful for future tasks.

We conduct our main evaluation on three LifelongAgentBench environments covering database interaction, operating-system tasks, and knowledge-graph reasoning, under both local and API-based deployments. Across all six environment–deployment combinations, TokenWorth achieves the highest task success rate among the evaluated methods while also using the fewest output tokens. These results show that its gains do not rely on longer generations; instead, selecting higher-value experience allows the frozen backbone to use a limited generation budget more effectively.

Our contributions are threefold:

- We formulate *resource-constrained lifelong agents* and study *resource robustness* under stringent joint constraints on generation, interaction, and memory.
- We propose **TokenWorth**, which learns query-dependent memory worth from online execution feedback and uses it consistently for memory retrieval and bounded memory management.
- We evaluate TokenWorth across three LifelongAgentBench environments under both local and API-based deployments, demonstrating improved task success and generation efficiency under stringent resource constraints.

2 Related Work

Lifelong Language Agents. Lifelong language agents aim to continually improve from accumulated interaction experience, enabling continued adaptation after deployment (Zheng et al., 2026; Fang et al., 2025). Early agent systems demonstrated that past trajectories, reflections, or reusable skills can improve future decision making, including Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), Voyager (Wang et al., 2023), and SocioDojo (Cheng & Chin, 2024). More recent work has developed explicit memory and skill mechanisms for continual adaptation, such as MemoryBank (Zhong et al., 2024), Mem0 (Chhikara et al., 2025), MemRL (Zhang et al., 2026c), MemSkill (Zhang et al., 2026a), and AutoSkill (Yang et al., 2026). Meanwhile, LifelongAgentBench (Zheng et al., 2025), AgentCL (Shu et al., 2026), SkillLearnBench (Zhong et al., 2026), and XSkill (Jiang et al., 2026) provide increasingly systematic evaluation of continual adaptation in language agents. Most of these studies focus on how agents can accumulate and exploit experience to improve capability; in contrast, we study whether such memory mechanisms remain effective under stringent joint constraints on generation, interaction, and memory.

Resource-Constrained Language Agents. Resource efficiency has been extensively studied for LLM inference and serving, including memory-efficient execution (Kwon et al., 2023) and collaborative inference under computation and communication constraints (Yuan et al., 2026). More recent work has begun to examine resource constraints at the agent level. Hajimiri et al. (2026) study whether online memory and skill modules remain beneficial under fixed inference-token budgets, while Wu et al. (2026) investigate continual learning with a hard-bounded on-device memory. BudgetMem (Zhang et al., 2026b) further considers query-aware routing across runtime memory budgets, and Chen & Cheng (2026) model agentic memory value using multiple factors for memory retention and retrieval.

These studies highlight the importance of accounting for resource cost in agent memory, but largely focus on individual resource constraints, runtime budgeting, or memory valuation in isolation. Our work instead studies *resource robustness* in lifelong adaptation under jointly constrained generation, interaction, and memory resources. We further identify memory-worth estimation as a central challenge in this setting and introduce **TokenWorth**, which learns query-dependent memory worth from streaming execution feedback and applies it consistently to both retrieval and bounded memory-bank management.

3 Resource-Constrained Lifelong Agents

We consider a lifelong agent that sequentially processes an online task stream $\mathcal{Q} = \{q_1, q_2, \ldots, q_T\}$. At step t, the agent receives task q_t and may retrieve a set of past experiences $\mathcal{R}_t$ from its current memory bank $\mathcal{B}_t = \{m_1, \ldots, m_{N_t}\}$. Conditioned on q_t and $\mathcal{R}_t$, the backbone model interacts with the environment for J_t rounds, producing a trajectory

$$\tau_t = \{(o_{t,j}, a_{t,j})\}_{j=1}^{J_t},$$

where $o_{t,j}$ and $a_{t,j}$ denote the observation and model output at interaction round j, respectively. The environment then returns a task-level reward $r_t \in \{0, 1\}$. After observing this outcome, the memory policy may update the bank from $\mathcal{B}_t$ to $\mathcal{B}_{t+1}$. In this work, we focus on a frozen-backbone setting, where continual adaptation is achieved solely through external memory.

We impose resource constraints on both task execution and experience reuse. For execution, the total amount of model generation associated with each task, including both environment interaction and any method-specific auxiliary language-model calls, is bounded by

$$\sum_{j=1}^{J_t} |a_{t,j}|_{\text{tok}} + A_t^{\text{aux}} \le O_{\max}, \qquad J_t \le J_{\max},$$

where A_t^{aux} denotes the output tokens generated by auxiliary language-model calls such as memory construction, reflection, or skill extraction. For memory, the persistent bank is bounded by

$$|\mathcal{B}_t| \le B_{\max},$$

while the memories supplied to the agent satisfy

$$|\mathcal{R}_t| \le K_{\max}, \qquad \mathrm{Len}(\mathcal{R}_t) \le L_{\max}.$$

We summarize the overall resource budget as

$$\mathcal{C} = (O_{\max}, J_{\max}, B_{\max}, K_{\max}, L_{\max}),$$

which jointly constrains generation, interaction, memory storage, and memory usage.

Let π_{mem} denote an online memory policy that determines both which experiences to retrieve for the current task and how the memory bank is updated after observing the task outcome. For a fixed resource budget $\mathcal{C}$, we define its lifelong task performance as

$$S(\pi_{\mathrm{mem}}; \mathcal{C}) = \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[r_t].$$

The objective is therefore

$$\pi^{\star}_{\mathrm{mem}} = \arg \max_{\pi_{\mathrm{mem}} \in \Pi(\mathcal{C})} S(\pi_{\mathrm{mem}}; \mathcal{C}), \tag{1}$$

where $\Pi(\mathcal{C})$ denotes the class of online memory policies satisfying the resource constraints above.

Our focus is on a stringent joint resource budget, under which generation, interaction, and memory resources are simultaneously limited. We use *resource robustness* to describe the ability of a memory mechanism to remain effective under such constraints. In this setting, an imperfect memory decision cannot easily be compensated for by retrieving additional experience, generating longer reasoning traces, or interacting for more rounds. Each retrieval and retention decision therefore becomes more consequential. The central challenge is not merely to identify which past experiences are relevant, but to determine which are actually *worth* allocating scarce memory resources to.

4 TokenWorth: Learning What to Remember

4.1 Overview

Figure 2 illustrates the overall workflow of **TokenWorth**. Given a query q_t, the agent first recalls a small candidate set from its bounded memory bank. TokenWorth then estimates the query-dependent worth of each candidate using a learnable linear base scorer together with an RFM-based residual correction, and selects the highest-valued experience for task execution. After receiving the environment reward, the worth model is updated online and the learned worth is further used to guide long-term memory-bank management.

TokenWorth consists of three components. First, an interpretable linear scorer captures stable global patterns of memory usefulness, while a residual Recursive Feature Machine (RFM) models nonlinear and query-specific corrections. Second, the residual learner separates fast value adaptation from slower AGOP-based feature adaptation for stable online learning. Third, the learned notion of worth guides both query-time retrieval and long-term memory management under a bounded memory budget.

4.2 Learning Query-Dependent Memory Worth

TokenWorth aims to determine which past experiences are worth allocating the agent's limited memory resources to. Unlike a fixed memory utility, the usefulness of an experience depends on the current query and must be inferred from sparse and indirect execution feedback. Each task is executed only once, feedback is observed only for the memory actually used, and an execution failure may arise from factors beyond the selected memory itself.

To balance stability and adaptivity under such feedback, TokenWorth decomposes memory worth into two complementary components. A lightweight linear base scorer captures stable and interpretable patterns that generalize across the task stream, while an RFM learns nonlinear residual structure not captured by the base scorer. The linear component is updated conservatively around an informative prior, whereas the RFM adapts more flexibly to local query–memory interactions.

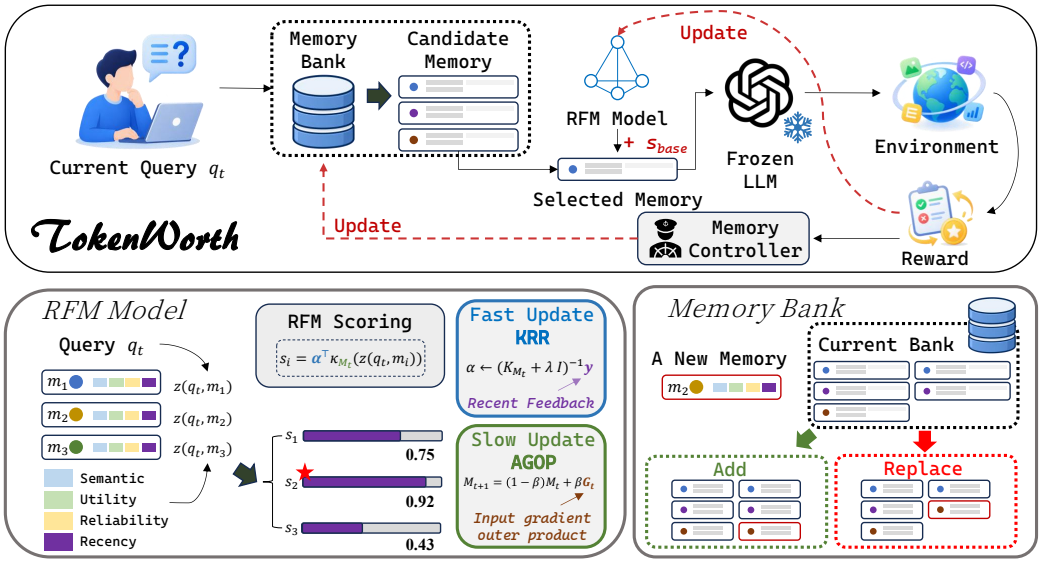


Figure 2: Overview of **TokenWorth**. TokenWorth learns query-dependent memory worth from online execution feedback and uses it for both memory retrieval and bounded memory-bank management.

Candidate recall and representation. Each memory $m_i \in \mathcal{B}_t$ contains a compact record of a previously successful interaction together with lightweight statistics describing its historical use. Given the current query q_t, we first recall a small candidate set using a lightweight textual relevance function:

$$\mathcal{C}_t = \mathrm{TopK}_{K_c} \{ s_{\mathrm{recall}}(q_t, m) : m \in \mathcal{B}_t \},$$

where K_c controls only the internal candidate pool and is independent of the number of memories ultimately supplied to the agent.

For each candidate $m_i \in \mathcal{C}_t$, we construct four interpretable features

$$\boldsymbol{\phi}_t^i = [\phi_{\mathrm{sim}}(q_t, m_i), \phi_{\mathrm{util}}(m_i), \phi_{\mathrm{rel}}(m_i), \phi_{\mathrm{recent}}(m_i)],$$

capturing semantic similarity, historical utility, reliability, and recency.

For residual estimation, we additionally construct a richer query–memory representation

$$\mathbf{z}_t^i = [|\mathbf{e}_t - \mathbf{e}_i|, \cos(\mathbf{e}_t, \mathbf{e}_i), u_i, \mathrm{rel}_t(m_i), \mathrm{rec}_t(m_i), \mathrm{freq}_t(m_i)].$$

This representation combines fine-grained semantic information with the empirical history of each memory. We use a lightweight deterministic text representation and maintain all memory statistics online; detailed definitions are provided in Appendix B.

Prior-regularized base scorer. The base scorer captures stable global relationships between memory characteristics and downstream usefulness:

$$s_{\mathrm{base},t}(q_t, m_i) = \mathbf{w}_t^\top \boldsymbol{\phi}_t^i.$$

Rather than learning $\mathbf{w}_t$ from scratch, we initialize it from an informative prior $\mathbf{w}_0$ and regularize subsequent online updates toward this prior. Given observed execution feedback $\mathcal{D}_t^{\mathrm{base}} = \{(\boldsymbol{\phi}_j^\star, y_j)\}$, we optimize

$$\mathcal{L}_{\mathrm{base}}(\mathbf{w}) = \sum_{(\boldsymbol{\phi}_j^\star, y_j) \in \mathcal{D}_t^{\mathrm{base}}} \log\big(1 + \exp(-y_j \mathbf{w}^\top \boldsymbol{\phi}_j^\star)\big) + \lambda_w \|\mathbf{w} - \mathbf{w}_0\|_2^2.$$

The prior anchors the scorer when evidence is scarce, while online updates allow it to gradually capture persistent patterns in the task stream. We further use conservative periodic updates, smoothing, and bounded weights to prevent unstable drift; implementation details are provided in Appendix B.

Residual worth estimation. The linear base scorer captures stable global regularities but cannot model all nonlinear or query-specific interactions among memory signals. TokenWorth therefore augments it with an RFM-based residual correction:

$$s_t(q_t, m_i) = s_{\text{base},t}(q_t, m_i) + \eta_t f_t(\mathbf{z}_t^i), \tag{2}$$

where f_t predicts the residual memory worth not explained by the base scorer and η_t controls its contribution. The selected memory is

$$m_t^\star = \arg\max_{m_i \in \mathcal{C}_t} s_t(q_t, m_i).$$

The two components therefore play complementary roles: the linear scorer captures stable global patterns of memory usefulness, while the RFM models nonlinear and query-specific deviations from these patterns.

After task execution, the observed reward provides execution-derived supervision for the selected query–memory pair:

$$y_t = 2r_t - 1 \in \{-1, +1\}.$$

The base scorer is updated directly from $(\boldsymbol{\phi}_t^\star, y_t)$. For the residual learner, we first compute the prediction implied by the base scorer,

$$\hat{y}_t^{\text{base}} = 2\sigma(s_{\text{base},t}(q_t, m_t^\star)) - 1,$$

and define the residual target

$$\delta_t = y_t - \hat{y}_t^{\text{base}}.$$

The RFM is then updated from $(\mathbf{z}_t^\star, \delta_t)$. Importantly, this supervision reflects downstream execution success rather than the counterfactual causal contribution of $m_t^\star$: TokenWorth does not re-execute the same task with alternative memories.

4.3 Two-Timescale RFM Adaptation

Within the residual learner, prediction and feature geometry need not adapt at the same rate. Recent execution outcomes provide immediate evidence about local residual structure, whereas the geometry governing query–memory features should evolve more conservatively. TokenWorth therefore separates *fast residual adaptation* from *slow geometry adaptation*.

Fast residual adaptation. Given a short feedback buffer $\mathcal{D}_t^{\text{fast}} = \{(\mathbf{z}_j, \delta_j)\}_{j=1}^{N_f}$, we fit a kernel ridge regressor

$$f_t(\mathbf{z}) = \sum_{j=1}^{N_f} \alpha_j K_{\mathbf{M}_t}(\mathbf{z}, \mathbf{z}_j), \qquad \boldsymbol{\alpha} = \left(\mathbf{K}_{\mathbf{M}_t} + \lambda \mathbf{I}\right)^{-1} \boldsymbol{\delta}, \tag{3}$$

where $\mathbf{M}_t$ is an adaptive diagonal feature metric. The KRR coefficients are updated frequently, allowing the residual learner to rapidly capture deviations from the global base scorer.

Slow geometry adaptation. A fixed kernel metric would assume that different query–memory features retain the same relative importance throughout the task stream. Instead, RFM adapts the feature geometry using the *Average Gradient Outer Product* (AGOP) (Radhakrishnan et al., 2024; Lai et al., 2026). Given a longer feedback window $\mathcal{D}_t^{\text{slow}} = \{(\mathbf{z}_j, \delta_j)\}_{j=1}^{N_s}$, we compute the diagonal AGOP

$$\widetilde{\mathbf{G}}_t = \text{Diag}\left(\frac{1}{N_s}\sum_{j=1}^{N_s} \nabla_{\mathbf{z}} f_t(\mathbf{z}_j) \odot \nabla_{\mathbf{z}} f_t(\mathbf{z}_j)\right),$$

and update the metric as

$$\mathbf{M}_{t+1} = \text{NormClip}\left((1-\beta)\mathbf{M}_t + \beta \widetilde{\mathbf{G}}_t\right). \tag{4}$$

After each metric update, the KRR predictor is refitted under the updated geometry.

The three adaptation mechanisms therefore play complementary roles. The prior-regularized linear scorer captures slowly evolving global regularities, the fast KRR update tracks local residual structure, and the slower AGOP update adapts the feature geometry used to represent these residuals.

4.4 WORTH-GUIDED MEMORY LIFECYCLE

Learning which memory to use for the current query addresses only part of the resource-allocation problem. Under a bounded persistent capacity $B_{\max}$, TokenWorth must also determine which experiences deserve to remain in memory for future tasks. We therefore extend the same query-dependent notion of worth from immediate retrieval to long-term memory management.

Only successful, externally verified interactions are considered as new memory candidates. For a candidate memory m, we estimate its broader worth over a small set of recent probe queries $\mathcal{P}_t$:

$$V_t(m) = \frac{1}{|\mathcal{P}_t|} \sum_{q \in \mathcal{P}_t} s_t(q, m). \tag{5}$$

While $s_t(q, m)$ measures the query-dependent worth of m for a particular query, $V_t(m)$ aggregates this signal over recent queries to estimate whether the memory deserves persistent capacity under the current task distribution.

TokenWorth follows a coverage–quality principle for memory admission. When capacity remains available, successful experiences are admitted to build initial coverage. Once the bank is full, a sufficiently novel memory may expand the represented experience space, whereas a redundant memory is retained only when its estimated worth improves upon a similar existing experience. Detailed novelty and replacement criteria are provided in Appendix B.

If an admitted candidate causes the bank to exceed its capacity, TokenWorth evicts the memory with the lowest probe-averaged worth:

$$m_{\text{evict}} = \arg \min_{m \in \mathcal{B}_t \cup \{m_{\text{new}}\}} V_t(m).$$

TokenWorth thus applies a unified notion of memory worth at two complementary timescales: query-dependent worth determines which experience should be used *now*, while probe-averaged worth determines which experience should be retained for *later*. This continually reallocates limited memory capacity toward experiences that remain useful for the evolving task stream. All adaptation occurs within the lightweight memory controller; TokenWorth requires no backbone updates, task replay, counterfactual execution, or additional language-model calls.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Benchmarks. We conduct our main evaluation on three environments from LifelongAgentBench (Zheng et al., 2025): database interaction (DB), operating-system interaction (OS), and knowledge-graph interaction (KG). These environments cover heterogeneous forms of interactive decision making while naturally following an online lifelong protocol, where experience accumulated from earlier tasks can be retained and reused in subsequent interactions. In all environments, the memory bank is initialized empty, each task is executed once, and memory decisions depend only on experience observed from previous tasks.

We additionally evaluate generalization beyond LifelongAgentBench using online lifelong variants of BigCodeBench (Zhuo et al., 2025) and ALFWorld (Shridhar et al., 2020). These results are reported in Appendix C.

Backbone and deployment. We evaluate TokenWorth under two complementary deployment settings. In the *local* setting, all methods use Qwen2.5-14B-Instruct (Qwen Team, 2024) as the frozen backbone, where generation and interaction are constrained by computation and latency. In the *API-based* setting, all methods use DeepSeek-V4-Flash (Xu et al., 2026) through an external API, where model generation and repeated interaction directly incur inference cost. No backbone parameters are updated throughout the task stream.

Resource-constrained protocol. Following Sec. 3, we evaluate all methods under a stringent joint resource budget that constrains generation, interaction, persistent memory capacity, and memory injection. Within each environment and deployment setting, all methods process the same task stream

Table 1: Task success rate (%) on LifelongAgentBench under stringent joint resource constraints. Best results are shown in **bold** and second-best results are underlined.

	Local: Qwen2.5-14B-Instruct			*API: DeepSeek-V4-Flash*		
Method	DB	OS	KG	DB	OS	KG
Standard	25.2	50.2	30.7	26.2	67.2	12.7
RAG	33.0	54.6	11.3	29.4	66.6	31.3
MemRL	28.0	52.2	4.7	30.4	66.2	32.0
Mem0	21.2	50.6	30.7	32.2	68.8	15.3
MemP	23.4	46.8	27.3	28.0	69.8	21.3
MemSkill	24.4	50.8	26.0	27.4	66.0	20.7
Reflexion	22.0	48.8	30.0	30.6	67.2	14.7
EvoSC	19.2	50.2	27.3	30.6	70.2	18.7
EvoSkill	24.6	49.6	30.0	25.6	68.4	14.7
TokenWorth	**65.0**	**59.4**	**36.7**	**62.2**	**73.2**	**36.7**

under the same resource budget. All language-model generation incurred by a method, including auxiliary calls for memory construction, reflection, or skill extraction, is counted toward the same per-task generation budget $O_{\max}$. This controlled protocol allows us to examine whether lifelong-memory mechanisms remain effective when both task execution and experience reuse are severely constrained. Detailed resource configurations are provided in the appendix.

Baselines. We compare TokenWorth with a no-memory agent (Standard), vanilla retrieval-augmented generation (RAG) (Lewis et al., 2020), and representative lifelong-memory methods, including MemRL (Zhang et al., 2026c), Mem0 (Chhikara et al., 2025), MemP (Fang et al., 2026), MemSkill (Zhang et al., 2026a), Reflexion (Shinn et al., 2023), EvoSC (Yu et al., 2026), and EvoSkill (Alzubi et al., 2026). All methods use the same frozen backbone, task stream, and resource budget within each evaluation setting.

Metrics. We evaluate task effectiveness using task success rate (SR $\uparrow$). To measure generation efficiency, we report the average number of output tokens per task (OutputTok $\downarrow$), including all language-model generation incurred by the method during both task execution and method-specific auxiliary operations such as memory construction, reflection, or skill extraction. We further report the relative change in output-token usage with respect to the no-memory Standard agent:

$$\Delta\mathrm{Tok} = \frac{\mathrm{OutputTok}_{\mathrm{method}} - \mathrm{OutputTok}_{\mathrm{Standard}}}{\mathrm{OutputTok}_{\mathrm{Standard}}} \times 100\%,$$

where a negative value indicates lower generation consumption than the no-memory baseline.

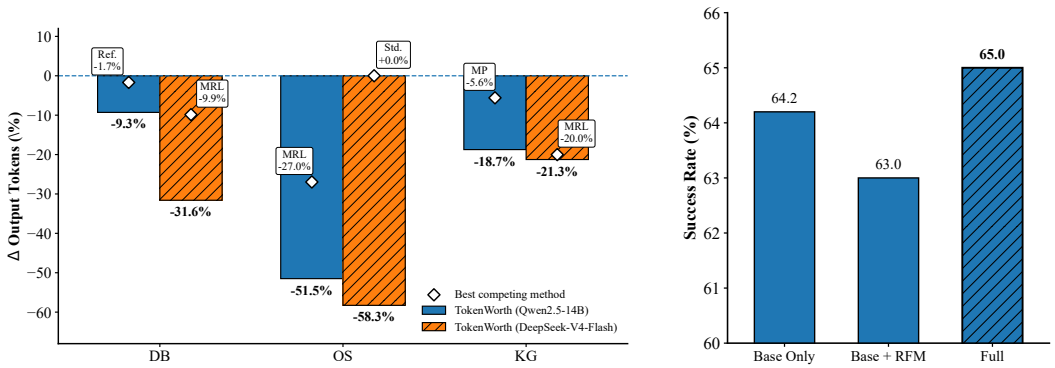
5.2 Results and Analysis

RQ1: Does TokenWorth remain effective under stringent resource constraints? Table 1 reports the main results on LifelongAgentBench. TokenWorth achieves the highest task success rate in all six environment–deployment combinations, indicating that its advantage is consistent across both local and API-based backbones.

Under the local deployment, the improvement is particularly pronounced on DB, where TokenWorth reaches 65.0%, outperforming the strongest baseline RAG (33.0%) by 32.0 percentage points. It also improves the strongest baseline by 4.8 points on OS and 6.0 points on KG. The same pattern holds under the API-based deployment: TokenWorth improves the best competing method by 30.0 points on DB, 3.0 points on OS, and 4.7 points on KG.

Notably, several memory-based methods provide limited gains over the no-memory Standard agent and can even substantially underperform it, particularly on KG. These results show that persistent memory alone does not guarantee effective lifelong adaptation under stringent resource constraints; the value of the experience selected for retention and reuse is critical.

RQ2: Does TokenWorth use the generation budget more effectively? Figure 3(a) compares TokenWorth's output-token usage across the six LifelongAgentBench environment–deployment com-



(a) Output-token efficiency

(b) Core component ablation

Figure 3: Efficiency and component analysis of TokenWorth. **(a)** Relative output-token usage on LifelongAgentBench with respect to the no-memory Standard agent; diamonds denote the most token-efficient competing method in each setting. **(b)** Core ablation on DB with Qwen2.5-14B-Instruct.

binations. We report the relative change with respect to the no-memory Standard agent, while the diamonds indicate the most token-efficient competing method. TokenWorth achieves the lowest output-token usage in all six settings.

Under the local Qwen2.5-14B-Instruct deployment, TokenWorth reduces output generation by 9.3%, 51.5%, and 18.7% on DB, OS, and KG, respectively. Under DeepSeek-V4-Flash, the corresponding reductions are 31.6%, 58.3%, and 21.3%. The reduction is particularly substantial on OS, where TokenWorth uses less than half the output tokens of the Standard agent under both deployments.

Importantly, these reductions are not obtained by sacrificing task performance. As shown in Table 1, TokenWorth simultaneously achieves the highest success rate in all six settings. It also remains more generation-efficient than the strongest competing method in each setting, with particularly clear margins on DB and OS. On API-based KG, the margin over MemRL is smaller, but TokenWorth still achieves both higher task success and lower output-token usage. Overall, the results indicate that TokenWorth's performance gains do not depend on longer generations: selecting higher-value experience allows the frozen backbone to solve more tasks while consuming less of the available generation budget.

RQ3: How do the core components affect memory-worth learning? Figure 3(b) reports a core ablation on DB with Qwen2.5-14B-Instruct. The base scorer alone achieves a strong success rate of 64.2%. Simply adding the RFM residual learner decreases performance to 63.0%, showing that a more expressive residual model does not automatically improve memory-worth estimation. The full TokenWorth framework reaches 65.0%, recovering the degradation and achieving the best performance.

This result suggests that TokenWorth's effectiveness does not come merely from adding a nonlinear predictor on top of the base scorer. Rather, the RFM residual must operate within the complete TokenWorth framework to provide stable and useful corrections to the base estimate.

6 Conclusion

We study lifelong language agents under stringent joint constraints on generation, interaction, and memory, and show that existing memory mechanisms can become ineffective in this setting. We propose **TokenWorth**, a lightweight online framework that learns query-dependent memory worth from execution feedback and uses it for both retrieval and bounded memory management. Across three LifelongAgentBench environments and both local and API-based deployments, TokenWorth achieves the highest task success while using the fewest output tokens, and additional experiments show competitive transfer to coding and embodied-agent tasks. Overall, our results suggest that effective lifelong adaptation under resource constraints depends not on remembering more, but on learning what is worth remembering.

AI USE STATEMENT

We used generative AI tools to assist with language editing, presentation refinement, and the preparation of plotting and analysis code. AI tools were also used to improve the clarity and organization of parts of the manuscript. All experimental design, implementation, execution, result verification, and scientific interpretation were performed and reviewed by the authors. We manually checked all AI-assisted text, code, figures, and analyses for correctness and consistency with the actual experiments. We take full responsibility for the final content of this work, including any text, code, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies resource-constrained lifelong language agents using existing research benchmarks and does not involve human subjects or the collection of personal or sensitive data. The proposed method operates on externally stored task experience and does not modify the underlying language model. We do not identify additional ethical concerns beyond those generally associated with the deployment of language-model-based agents.

REPRODUCIBILITY STATEMENT

We provide detailed implementation and experimental information in the appendix, including the online learning protocol, memory representation and management procedure, resource-budget configurations, optimization and update settings, and additional experimental details. These descriptions are intended to make the TokenWorth implementation and the reported evaluation protocol reproducible.

REFERENCES

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

Zhibao Chen and Qian Cheng. Learning what to remember: A cognitively grounded multi-factor value model for agentic memory. *arXiv preprint arXiv:2606.12945*, 2026.

Junyan Cheng and Peter Chin. Sociodojo: Building lifelong analytical agents with real-world text and time series. In *ICLR*, 2024.

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.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, et al. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, 2026.

Shubham Gupta. Ai agents collaboration under resource constraints: Practical implementations. *International journal of artificial intelligence research and development*, 3(1):51–63, 2025.

Sina Hajimiri, Masih Aminbeidokhti, Jose Dolz, Ismail Ben Ayed, Issam H Laradji, Spandana Gella, and Nicolas Gontier. Are online skill and memory modules always worth their tokens? a budget-constrained study of web agents. *arXiv preprint arXiv:2606.15017*, 2026.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R Fung. Xskill: Continual learning from experience and skills in multimodal agents. In *ICML*, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Guannan Lai, Da-Wei Zhou, Zhenguo Li, and Han-Jia Ye. The golden subspace: Where efficiency meets generalization in continual test-time adaptation. In *CVPR*, 2026.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *NeurIPS*, 2020.

Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. Agrail: A lifelong agent guardrail with effective and adaptive safety detection. In *ACL*, 2025.

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

Adityanarayanan Radhakrishnan, Daniel Beaglehole, Parthe Pandit, and Mikhail Belkin. Mechanism for feature learning in neural networks and backpropagation-free machine learning models. *Science*, 383(6690):1461–1467, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *NeurIPS*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Yiheng Shu, Bernal Jiménez Gutiérrez, Saisri Padmaja Jonnalagedda, Yuguang Yao, Huan Sun, and Yu Su. Agentcl: Toward rigorous evaluation of continual learning in language agents. *arXiv preprint arXiv:2606.02461*, 2026.

Xu Sun, Tongkai Xu, Baiheng Xie, Li Huang, Qiang Gao, and Kunpeng Zhang. M-rag: Semantic key-value indexing for retrieval-augmented generation. *arXiv e-prints*, pp. arXiv–2603, 2026a.

Xun Sun, Baiheng Xie, Li Huang, and Qiang Gao. Scaling dpps for rag: Density meets diversity. *arXiv preprint arXiv:2604.03240*, 2026b.

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

Beining Wu, Zihao Ding, Jun Huang, and Yanxiao Zhao. Forget to improve: On-device llm-agent continual learning via budget-curated memory. *arXiv preprint arXiv:2606.25115*, 2026.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, et al. Autoskill: Experience-driven lifelong learning via skill self-evolution. *arXiv preprint arXiv:2603.01145*, 2026.

Hongzhuo Yu, Fei Zhu, Guo-Sen Xie, and Ling Shao. Self-consolidation for self-evolving agents. *arXiv preprint arXiv:2602.01966*, 2026.

Liangqi Yuan, Wenzhi Fang, Shiqiang Wang, H Vincent Poor, and Christopher G Brinton. Large language models over networks: Collaborative intelligence under resource constraints. *arXiv preprint arXiv:2605.08626*, 2026.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026a.

Haozhen Zhang, Haodong Yue, Tao Feng, Quanyu Long, Jianzhu Bao, Bowen Jin, Weizhi Zhang, Xiao Li, Jiaxuan You, Chengwei Qin, et al. Learning query-aware budget-tier routing for runtime agent memory. *arXiv preprint arXiv:2602.06025*, 2026b.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, et al. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026c.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *AAAI*, 2024.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. Lifelongagentbench: Evaluating llm agents as lifelong learners. *arXiv preprint arXiv:2505.11942*, 2025.

Junhao Zheng, Chengming Shi, Xidi Cai, Qiuke Li, Duzhen Zhang, Chenxing Li, Dong Yu, and Qianli Ma. Lifelong learning of large language model based agents: A roadmap. *TPAMI*, 2026.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo FR Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. Skilllearnbench: Benchmarking continual learning methods for agent skill generation on real-world tasks. *arXiv preprint arXiv:2604.20087*, 2026.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *AAAI*, 2024.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, volume 2025, pp. 66602–66656, 2025.

A Additional Experimental Details

A.1 Benchmarks

Our main experiments use three environments from LifelongAgentBench (Zheng et al., 2025): database interaction (DB), operating-system interaction (OS), and knowledge-graph interaction (KG). All three environments follow an online lifelong protocol in which tasks are processed sequentially and experience accumulated from earlier tasks may be retained and reused in subsequent interactions.

The environments cover substantially different forms of interactive reasoning. DB requires the agent to interact with a database through structured operations and answer questions based on the resulting observations. OS consists of system-management tasks that require multi-step interaction through Bash commands. KG follows an AgentBench-style interaction protocol in which the agent iteratively issues SPARQL queries to a knowledge graph before producing a final answer. Together, these environments provide heterogeneous lifelong task streams for evaluating memory mechanisms under constrained generation, interaction, and memory resources.

We additionally evaluate generalization beyond LifelongAgentBench on BigCodeBench (Zhuo et al., 2025) and ALFWorld (Shridhar et al., 2020). Their construction and results are reported separately in Appendix C.

A.2 Baselines

We compare TokenWorth with a no-memory agent (Standard), vanilla retrieval-augmented generation (RAG) (Lewis et al., 2020), and a broad set of representative lifelong-memory and experience-learning methods.

Standard. Standard uses the frozen backbone without persistent cross-task memory. Each task is solved independently using only the current task context and environment observations. It serves as the no-memory reference for measuring whether accumulated experience provides a net benefit.

RAG. RAG (Lewis et al., 2020; Sun et al., 2026a;b) stores successful historical experiences and retrieves memories according to textual relevance to the current query. It represents a lightweight similarity-based memory mechanism without online value learning.

MemRL. MemRL (Zhang et al., 2026c) learns runtime utility signals for episodic memories from downstream task outcomes, enabling experience selection beyond pure semantic similarity. It serves as a value-aware memory baseline for evaluating TokenWorth's query-dependent memory-worth estimation.

Mem0. Mem0 (Chhikara et al., 2025) provides a long-term memory architecture for LLM agents with mechanisms for memory extraction, storage, retrieval, and update. We adapt it to the same online task stream and resource constraints as the other methods.

MemP. MemP (Fang et al., 2026) maintains persistent task experience and reuses stored information in subsequent tasks.

MemSkill. MemSkill (Zhang et al., 2026a) transforms agent experience into reusable memory skills that can be retained and reused across future tasks.

Reflexion. Reflexion (Shinn et al., 2023) converts task feedback into verbal reflections that are stored and reused in later interactions, enabling cross-task adaptation without updating the backbone model.

EvoSC. EvoSC (Yu et al., 2026) continually constructs and evolves reusable experience from agent interactions.

EvoSkill. EvoSkill (Alzubi et al., 2026) extracts reusable skills from previous task trajectories and continually updates its skill repository as additional experience becomes available.

Within each environment and deployment setting, all methods use the same frozen backbone, task sequence, environment interface, and resource budget whenever applicable. If a baseline requires additional language-model calls for memory construction, reflection, or skill extraction, we retain these operations as specified by the corresponding method rather than removing them for efficiency. All output tokens generated by these auxiliary calls are counted toward the same generation budget and are included in the reported OutputTok.

A.3 Backbones and Deployment Settings

We evaluate all methods under two complementary deployment settings.

Local deployment. The local setting uses Qwen2.5-14B-Instruct (Qwen Team, 2024) as the frozen backbone and runs model inference on NVIDIA A800 GPUs. The backbone parameters remain fixed throughout the entire lifelong task stream.

API-based deployment. The API-based setting uses DeepSeek-V4-Flash (Xu et al., 2026) as the frozen backbone through an external inference API. The same online protocol is used, but model generation and repeated interaction directly incur inference cost through API usage.

For both deployment settings, all methods process the same task sequence within each environment. Memory banks are initialized empty and evolve only from information observed earlier in the online stream. No backbone parameters are updated. TokenWorth additionally requires no offline pretraining, task replay, or counterfactual re-execution of the same task.

A.4 Evaluation Protocol

Our primary effectiveness metric is task success rate (SR $\uparrow$). A task is considered successful only when the environment evaluator determines that the final answer or environment state is correct.

To evaluate generation efficiency, we report the average number of output tokens per task (OutputTok $\downarrow$). OutputTok includes all language-model-generated tokens incurred by a method, covering both task execution and any auxiliary language-model calls used for memory construction, reflection, skill extraction, or other method-specific processing. We further compute the relative change in output-token usage with respect to the no-memory Standard agent:

$$\Delta\mathrm{Tok} = \frac{\mathrm{OutputTok}_{\mathrm{method}} - \mathrm{OutputTok}_{\mathrm{Standard}}}{\mathrm{OutputTok}_{\mathrm{Standard}}} \times 100\%.$$

Negative values therefore indicate lower generation consumption than the no-memory reference.

Where relevant, we additionally record wall-clock runtime and agent-side memory usage as supplementary system statistics. Runtime covers the complete online task stream, including memory construction, retrieval, and memory-controller updates. Memory usage is measured using average and peak resident set size and is reported in MiB. These system statistics are supplementary to the primary task-success and generation-efficiency metrics used in the main evaluation.

B Implementation Details of TokenWorth

B.1 Common Online Implementation

TokenWorth follows a strictly online lifelong-learning protocol. At step t, all retrieval and memory-management decisions depend only on the current memory bank and feedback observed from tasks up to step t. Each task is executed once; we do not perform offline pretraining, task replay, or counterfactual execution with alternative memories. The backbone model remains frozen throughout the entire task stream.

Lightweight text representation. We use a deterministic 512-dimensional text representation constructed from token hashing and character 3–5-gram hashing, followed by ℓ_2 normalization. The representation requires neither additional language-model calls nor gradient-based encoder training.

For each query–memory pair, TokenWorth constructs the residual feature representation

$$\mathbf{z}(q, m) = [|\mathbf{e}_q - \mathbf{e}_m|, \cos(\mathbf{e}_q, \mathbf{e}_m), u_m, \mathrm{rel}(m), \mathrm{rec}(m), \mathrm{freq}(m)] .$$

Here, u_m denotes historical utility. Reliability is defined as

$$\mathrm{rel}(m) = \frac{h_m + 1}{h_m + d_m + 2},$$

where h_m and d_m are the numbers of helpful and harmful uses, respectively. Recency is

$$\mathrm{rec}(m) = \exp[-\kappa \Delta t_m],$$

where Δt_m is the number of steps since the memory was last accessed. Access frequency is normalized by memory lifetime:

$$\mathrm{freq}(m) = \frac{c_m}{\max(1, t - \tau_m + 1)},$$

where c_m is the access count and τ_m is the creation step. We use $\kappa = 0.02$ throughout the main experiments.

Candidate recall. Before learned ranking, TokenWorth uses lightweight textual relevance to reduce the bounded memory bank to a small candidate set:

$$\mathcal{C}_t = \mathrm{TopK}_{K_c} \{ s_{\mathrm{recall}}(q_t, m) : m \in \mathcal{B}_t \} .$$

This stage serves only as inexpensive candidate screening. All subsequent memory selection is performed by the learned worth model described below.

Prior-regularized base scorer. For each candidate memory, we construct four interpretable features

$$\boldsymbol{\phi}(q, m) = [\phi_{\mathrm{sim}}(q, m), \phi_{\mathrm{util}}(m), \phi_{\mathrm{rel}}(m), \phi_{\mathrm{recent}}(m)] .$$

The base score is

$$s_{\mathrm{base},t}(q, m) = \mathbf{w}_t^\top \boldsymbol{\phi}(q, m).$$

Rather than learning $\mathbf{w}_t$ from scratch under sparse online feedback, we initialize it from

$$\mathbf{w}_0 = [1.0,\ 0.10,\ 0.10,\ 0.02]$$

and regularize online updates toward this prior. The base-feedback set $\mathcal{D}_t^{\mathrm{base}}$ is maintained as a FIFO sliding window containing at most 256 observed query–memory feedback pairs. Given the current buffer, we optimize

$$\mathcal{L}_{\mathrm{base}}(\mathbf{w}) = \sum_{(\boldsymbol{\phi}_j, y_j) \in \mathcal{D}_t^{\mathrm{base}}} \log \left(1 + \exp[-y_j \mathbf{w}^\top \boldsymbol{\phi}_j]\right) + \lambda_w \|\mathbf{w} - \mathbf{w}_0\|_2^2.$$

We use $\lambda_w = 0.1$. The base scorer is refitted after every 50 newly observed feedback examples. To reduce abrupt changes under sparse online supervision, the newly fitted weights are combined with the previous weights using an exponential moving average with smoothing coefficient $\beta_w = 0.5$.

After smoothing, the weights are clipped coordinate-wise to

$$\mathbf{w}_{\min} = [0,\ 0,\ 0,\ 0], \qquad \mathbf{w}_{\max} = [2.0,\ 1.0,\ 1.0,\ 1.0].$$

This keeps the interpretable feature contributions bounded and prevents unstable drift during online adaptation.

Residual memory-worth estimation. TokenWorth combines the global base score with an RFM-based residual:

$$s_t(q, m) = s_{\text{base},t}(q, m) + \eta_t f_t(\mathbf{z}(q, m)).$$

The selected memory is

$$m_t^\star = \arg\max_{m \in \mathcal{C}_t} s_t(q_t, m).$$

The coefficient η_t controls the influence of the residual component and is gradually introduced during early online adaptation. Environment-specific values are reported in Table A1.

After observing task reward $r_t \in \{0, 1\}$, we define

$$y_t = 2r_t - 1.$$

The base scorer is updated using $(\boldsymbol{\phi}_t^\star, y_t)$. For the residual learner, we first convert the base score into an outcome-scale prediction,

$$\widehat{y}_t^{\text{base}} = 2\sigma\left(s_{\text{base},t}(q_t, m_t^\star)\right) - 1,$$

and construct the residual target

$$\delta_t = y_t - \widehat{y}_t^{\text{base}}.$$

The RFM therefore learns execution-derived residual structure that remains unexplained by the linear base scorer.

Kernel ridge regression. The RFM residual predictor uses kernel ridge regression with a Laplacian kernel under an adaptive diagonal metric:

$$K_{\mathbf{M}_t}(\mathbf{z}, \mathbf{z}') = \exp\left(-\gamma\sqrt{(\mathbf{z} - \mathbf{z}')^\top \mathbf{M}_t(\mathbf{z} - \mathbf{z}')}\right).$$

Given a bounded recent-feedback buffer $\{(\mathbf{z}_j, \delta_j)\}_{j=1}^{N_f}$, the KRR predictor is fitted to the residual targets. The fast feedback buffer contains at most 128 examples. We use KRR ridge regularization $\lambda_{\text{KRR}} = 0.01$ and kernel parameter $\gamma = 1.0$.

AGOP adaptation. Feature geometry adapts on a slower timescale than the KRR coefficients. Given

$$\mathbf{g}_j = \nabla_{\mathbf{z}} f_t(\mathbf{z}_j),$$

we compute the diagonal Average Gradient Outer Product (AGOP)

$$\widetilde{\mathbf{G}}_t = \text{Diag}\left(\frac{1}{N_s}\sum_{j=1}^{N_s} \mathbf{g}_j \odot \mathbf{g}_j\right).$$

The slower AGOP feedback buffer contains at most 256 examples. The feature metric is updated as

$$\mathbf{M}_{t+1} = \text{NormClip}\left((1 - \beta_M)\mathbf{M}_t + \beta_M \widetilde{\mathbf{G}}_t\right),$$

with $\beta_M = 0.25$. Each diagonal weight is clipped to $[0.05, 8.0]$, after which the metric is normalized to have mean weight 1. Following every AGOP update, the KRR predictor is immediately refitted under the updated geometry.

Execution-derived memory statistics. Only the memory actually selected for task execution receives online feedback. Each newly created memory is initialized with utility

$$u(m) = 1.0.$$

Unlike the base and AGOP updates, memory utility does not use an exponential moving average. Instead, utility is updated additively from the observed task outcome:

$$u_t(m_t^\star) = \text{clip}\left(u_{t-1}(m_t^\star) + \begin{cases} 0.25, & r_t = 1, \\ -0.45, & r_t = 0, \end{cases} -2,\ 5\right).$$

Thus, successful use increases the utility by 0.25, whereas unsuccessful use decreases it by 0.45, with utility always bounded to $[-2, 5]$. A successful use also increments the helpful-use count h_m, whereas a failed use increments the harmful-use count d_m. The access count and last-access step are updated after every use.

Importantly, all of these signals are derived from observed downstream execution. TokenWorth does not estimate the counterfactual contribution of unselected memories by re-executing the same task.

Memory admission and replacement. Only successful, externally verified interactions are eligible to create new persistent memories. When

$$|\mathcal{B}_t| < B_{\max},$$

a valid new memory is admitted directly to establish initial coverage.

Once the bank reaches capacity, TokenWorth jointly considers coverage and learned worth. For a candidate m_{new}, novelty is measured as

$$\text{nov}_t(m_{\text{new}}) = 1 - \max_{m_i \in \mathcal{B}_t} \cos(\mathbf{e}_{\text{new}}, \mathbf{e}_i).$$

We use a novelty threshold

$$\tau_{\text{nov}} = 0.35.$$

To estimate whether a memory remains useful under the recent task distribution, we maintain a short FIFO history containing at most eight previous queries on which memory retrieval was performed. The probe set $\mathcal{P}_t$ consists of the current query together with these recent retrieval queries, giving

$$|\mathcal{P}_t| \leq 9.$$

The persistent value of memory m is estimated using the same learned worth score employed at retrieval time:

$$V_t(m) = \frac{1}{|\mathcal{P}_t|} \sum_{q \in \mathcal{P}_t} s_t(q, m).$$

The memory controller uses an improvement margin

$$\delta_{\text{write}} = 0.02$$

when determining whether a candidate provides a meaningful probe-based value improvement over already represented experience. It additionally uses a positive-probe-ratio threshold

$$\tau_{\text{pos}} = 0.45$$

as a consistency criterion over the probe set. These criteria complement the novelty test so that memory admission is not determined by semantic novelty alone.

Let m_{near} denote the most similar memory already in the bank. A nearest memory is considered eligible for direct replacement only when

$$u(m_{\text{near}}) < 0.5.$$

For such replacement candidates, TokenWorth additionally requires

$$V_t(m_{\text{new}}) > V_t(m_{\text{near}}) + \delta_{\text{rep}},$$

where

$$\delta_{\text{rep}} = 0.10.$$

This conservative replacement rule prevents a newly observed experience from displacing an existing memory unless the existing memory has low historical utility and the candidate provides a clear probe-based value advantage.

Value-based eviction. If admitting a candidate temporarily exceeds the persistent-memory budget, TokenWorth evicts the memory with the lowest probe-averaged worth:

$$m_{\text{evict}} = \arg \min_{m \in \mathcal{B}_t \cup \{m_{\text{new}}\}} V_t(m).$$

The resulting bank therefore always satisfies

$$|\mathcal{B}_{t+1}| \leq B_{\max}.$$

Table A1: Environment-specific resource and adaptation configurations used in the main LifelongAgentBench experiments.

Setting	DB	OS	KG
Generation budget $O_{\max}$	128	512	128
Interaction budget $J_{\max}$	3	5	15
Memory capacity $B_{\max}$	64	64	64
Retrieved memories $K_{\max}$	1	1	1
Injection budget $L_{\max}$	900 chars	800 chars	4096 chars
Candidate pool size K_c	12	12	12
KRR update interval	50	25	25
AGOP update interval	200	100	100
Residual influence η	0.15	0.05	0.10
RFM warm-up feedback	4	4	24

B.2 Common Hyperparameters

Unless otherwise specified, all environments use the same lightweight online learning and memory-management configuration.

For the base scorer, we use

$$\mathbf{w}_0 = [1.0, 0.10, 0.10, 0.02], \qquad \lambda_w = 0.1,$$

with a FIFO feedback buffer of at most 256 examples. The base scorer is refitted after every 50 newly collected feedback examples and the updated weights are smoothed using $\beta_w = 0.5$. The coordinate-wise lower and upper bounds are

$$\mathbf{w}_{\min} = [0, 0, 0, 0], \qquad \mathbf{w}_{\max} = [2.0, 1.0, 1.0, 1.0].$$

For the residual learner, the fast KRR buffer contains at most 128 feedback examples and uses

$$\lambda_{\mathrm{KRR}} = 0.01, \qquad \gamma = 1.0.$$

The slower AGOP buffer contains at most 256 feedback examples. The AGOP metric uses smoothing coefficient $\beta_M = 0.25$, clips each diagonal weight to $[0.05, 8.0]$, and subsequently normalizes the metric to have mean diagonal weight 1.

For execution-derived memory statistics, newly created memories have initial utility 1.0. Successful uses add 0.25 and unsuccessful uses subtract 0.45, with utility clipped to $[-2, 5]$. Memory management uses

$$\tau_{\mathrm{nov}} = 0.35, \qquad \delta_{\mathrm{write}} = 0.02, \qquad \tau_{\mathrm{pos}} = 0.45, \qquad \delta_{\mathrm{rep}} = 0.10.$$

The probe history is a FIFO queue containing at most eight previous queries with memory retrieval; including the current query, the probe set therefore contains at most nine queries. A nearest existing memory is eligible for direct replacement only when its historical utility is below 0.5.

B.3 Environment-Specific Configuration

TokenWorth uses the same memory-worth learning mechanism across DB, OS, and KG, while resource budgets and online adaptation schedules are adjusted to the interaction characteristics of each environment. Table A1 summarizes the configurations used in the main experiments.

B.4 Environment-Specific Memory Construction

DB. For DB, a successful interaction is stored as a compact demonstration containing the original question, valid SQL actions, truncated database observations, and the final answer. Invalid actions and unnecessary environment output are removed before storage. At retrieval time, TokenWorth first recalls up to $K_c = 12$ candidates using textual relevance and then selects one experience using the learned memory-worth score.

Table A2: Task success rate (%) on additional generalization benchmarks. Best results are shown in **bold** and second-best results are underlined.

	Local: Qwen2.5-14B-Instruct		*API: DeepSeek-V4-Flash*
Method	BIGCODE	ALFWORLD	BIGCODE
Standard	41.7	59.1	49.9
RAG	42.6	73.2	**50.9**
MemRL	42.9	69.7	50.5
Mem0	41.7	64.1	50.2
MemP	42.3	66.4	49.6
MemSkill	42.9	63.6	49.1
Reflexion	40.8	69.2	49.0
EvoSC	42.6	65.0	49.6
EvoSkill	41.8	64.8	48.8
TokenWorth	**43.1**	**73.5**	50.6

OS. Complete shell trajectories often contain instance-specific details such as paths, user names, and process identifiers that transfer poorly across tasks. For successful OS interactions, we therefore retain compact reusable execution patterns derived from the successful Bash actions, including common operations for permission management, batch file processing, text editing, symbolic links, and process management.

A task is treated as successful whenever the environment evaluator reports `correct`, including cases in which the interaction-round limit is reached after the environment has already entered the correct final state. As in DB, TokenWorth first recalls up to 12 candidate memories and then selects a single experience for injection using the learned worth score.

KG. KG follows the AgentBench-style SPARQL interaction protocol. Successful memories retain useful SPARQL actions together with truncated knowledge-graph observations. Because KG typically requires longer multi-step interaction, we allow up to 15 interaction rounds and an injection budget of 4096 characters. To reduce instability during early online learning, residual adaptation begins only after at least 24 query–memory feedback pairs have been collected.

C GENERALIZATION BEYOND LIFELONGAGENTBENCH

To evaluate whether TokenWorth transfers beyond the interactive environments used in our main experiments, we additionally consider online lifelong variants of BigCodeBench (Zhuo et al., 2025) and ALFWorld (Shridhar et al., 2020). BigCodeBench evaluates code generation, whereas ALFWorld evaluates embodied decision making in text-based household environments. In both settings, tasks are processed sequentially, the memory bank is initialized empty, and memory decisions depend only on experience observed from preceding tasks.

Table A2 reports the additional generalization results. Under the local deployment, TokenWorth achieves the highest success rate on both benchmarks, reaching 43.1% on BigCodeBench and 73.5% on ALFWorld. The margins over the strongest competing methods are small: 0.2 percentage points on BigCodeBench and 0.3 points on ALFWorld.

Under the API-based BigCodeBench setting, TokenWorth achieves 50.6%, compared with 50.9% for RAG. Thus, unlike the larger improvements observed on the DB, OS, and KG environments in the main evaluation, TokenWorth is broadly competitive rather than consistently superior on these additional task domains.

These results indicate that the memory-worth mechanism transfers to coding and embodied-agent settings without modifying the frozen backbone, while also showing that its largest gains arise in the interactive lifelong environments studied in the main experiments.

Table A3: Hyperparameter sensitivity of TokenWorth on DB with Qwen2.5-14B-Instruct. The full configuration achieves 65.0% accuracy. Δ denotes the accuracy change in percentage points relative to the full configuration.

Parameter	Default	Tested value	Accuracy (%)	Δ (pp) [CI]
RFM influence η	0.15	0.05	62.4	-2.6 [-4.8, -0.4]
		0.10	63.8	-1.2 [-3.4, $+0.8$]
		0.20	64.2	-0.8 [-3.2, $+1.6$]
		0.30	64.0	-1.0 [-3.2, $+1.2$]
Base regularization λ_w	0.10	0.03	64.8	-0.2 [-2.2, $+2.0$]
		0.30	62.6	-2.4 [-5.0, 0.0]
		1.00	63.0	-2.0 [-4.4, $+0.2$]
Base smoothing	0.50	0.25	64.2	-0.8 [-3.2, $+1.4$]
		0.75	62.8	-2.2 [-4.8, $+0.2$]
		1.00	62.6	-2.4 [-4.6, -0.2]
Base update interval	50	25	65.2	$+0.2$ [-1.6, $+2.2$]
		100	62.4	-2.6 [-5.0, -0.2]
AGOP smoothing	0.25	0.10	64.4	-0.6 [-2.8, $+1.4$]
		0.50	64.6	-0.4 [-2.4, $+1.6$]
		0.75	62.2	-2.8 [-4.8, -0.8]
KRR update interval	50	25	62.2	-2.8 [-5.2, -0.4]
		100	63.4	-1.6 [-4.0, $+0.6$]
AGOP update interval	200	100	62.6	-2.4 [-4.4, -0.4]
		400	62.0	-3.0 [-5.2, -0.8]
Kernel γ	1.0	0.5	64.0	-1.0 [-3.4, $+1.4$]
		2.0	63.0	-2.0 [-4.4, $+0.4$]

D HYPERPARAMETER SENSITIVITY

We further evaluate the sensitivity of TokenWorth to its main online adaptation hyperparameters. All experiments are conducted on DB with Qwen2.5-14B-Instruct using the same 500-task stream as the corresponding main experiment. Starting from the default TokenWorth configuration, we vary one parameter at a time while keeping all remaining settings fixed. The full configuration achieves 65.0% task success rate.

Table A3 reports the resulting accuracy, the change relative to the full configuration, and the corresponding confidence interval for the paired accuracy difference.

TokenWorth is relatively insensitive to moderate perturbations of several key parameters. For example, varying the RFM influence from 0.15 to values between 0.10 and 0.30, changing AGOP smoothing from 0.25 to 0.10 or 0.50, and varying the kernel parameter γ produce only modest changes, with the reported confidence intervals overlapping zero.

Larger changes to the online adaptation dynamics lead to clearer degradation. Reducing the RFM influence to 0.05 decreases accuracy by 2.6 points. Removing effective base-score smoothing (1.0), slowing the base update interval to 100, and increasing AGOP smoothing to 0.75 reduce accuracy by 2.4–2.8 points. The adaptation schedules are also important: updating KRR every 25 feedback examples reduces accuracy by 2.8 points, while changing the AGOP update interval from the default 200 to 100 or 400 decreases accuracy by 2.4 and 3.0 points, respectively.

Overall, the results indicate that TokenWorth does not depend on a narrowly tuned value for most individual hyperparameters, but is more sensitive to aggressive changes in the update and smoothing schedules. This observation is consistent with the motivation for conservative, two-timescale online adaptation used in the full method.