TINYACE: Bounded Working-Memory Context Engineering for Small Language Models (SLMs)

Suryodaya B. Shahi[1*], **Sathwik H. Naik**[1], **Archit Harsh**[1]
[1]University of Maryland, College Park
{sshahi20, sathwikh, aharsh}@umd.edu

Abstract

Self-improvement frameworks such as *Agentic Context Engineering* (ACE) maintain an evolving playbook of lessons that are retrieved and integrated into the prompt across queries [4, 8]. While effective at larger scales, this mechanism is brittle for Small Language Models (SLMs) with limited context windows and strict latency constraints. We present TINYACE, a lightweight adaptation of ACE that introduces a **bounded working-memory playbook** and **strategic forgetting** to keep context within a fixed token budget.

We evaluate TINYACE on a fixed sequential slice (n=50) of the **SciQ** test split [6] across three model scales (1.1B, 3.8B, 7B) on a single NVIDIA A100. Our results identify a **"Capacity Sweet Spot"**: TINYACE improves mid-sized edge-scale models (Phi-3 Mini 3.8B) by **+4% Accuracy** (37/50 $\rightarrow$ 39/50), while degrading very small models (TinyLlama 1.1B) and providing limited benefit when a strong model already saturates the benchmark (Mistral 7B, 96% baseline). A detailed ablation study further reveals that **failure tracking is the most critical component** for accuracy, and that simple **FIFO eviction often outperforms complex scoring** for SLMs.

1 Introduction

Agentic self-improvement methods iteratively enrich a model's prompt with distilled lessons from previous attempts [8]. This approach is attractive for Small Language Models (SLMs, 1B–7B parameters) running under edge constraints. However, naive accumulation of lessons leads to **context saturation**: prompt growth increases latency and may confuse smaller models with stale advice. Naive accumulation can cause prompt drift, where instructions become longer and less specific over time, eventually eroding the model's adherence to the original task.

Prior systems such as MemGPT [3] propose hierarchical memory, while KV-cache eviction works [5] target token-level efficiency. Our focus is distinctly on **lightweight semantic memory**—managing specific reasoning strategies within a strict token budget.

Contribution. We present TINYACE, a bounded working-memory (WM) playbook that caps context size (e.g., 512 tokens). We utilize a feedback loop that updates memory primarily on failures to prevent redundancy. We provide:

1. **A Multi-Scale Evaluation:** Testing across 1.1B, 3.8B, and 7B models.
2. **Ablation Study:** Isolating the impact of recency bias, failure tracking, and vagueness detection.
3. **Design Insights:** Demonstrating that simple FIFO eviction provides a strong baseline for SLMs.

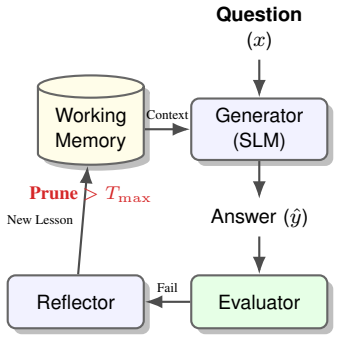

Figure 1: TINYACE Architecture. Lessons are generated primarily on failures; the playbook is pruned when exceeding $T_{\max}$.

2 Method: TINYACE

2.1 Architecture

The TINYACE architecture separates the **Generator** (the SLM) from its **Working Memory** (Playbook). As shown in Figure 1, the system operates in a loop:

- **Evaluation Loop:** The system checks the Generator's answer against ground truth.
- **Reflection:** Lessons are generated primarily when the model fails, with optional periodic updates on correct answers to reinforce success.
- **Pruning:** Before insertion, if the Playbook exceeds the token budget $T_{\max}$, the **Eviction Policy** removes entries.

2.2 Bounded Eviction Policies

When the playbook exceeds $T_{\max}$, TINYACE supports multiple eviction strategies. In this work, we report results using

[*]S.Shahi et al

Utility Scoring (Eq. 1) for the main results, and compare it against **FIFO** in our ablation study.

The retention score $S(e)$ for a strategy e is defined as:

$$S(e) = \alpha \frac{N_{\text{succ}}}{N_{\text{used}} + \epsilon} - \beta \frac{N_{\text{fail}}}{N_{\text{used}} + \epsilon} + \gamma e^{-\lambda(t - t_{\text{last}})} - \delta \mathcal{V}(e) \quad (1)$$

where t is the current step, t_{last} is the last usage step, ϵ is a smoothing constant, and $\mathcal{V}(e)$ penalizes vague strategies.

3 Experimental Setup

Dataset. SciQ [6] Test split. We report results on the first n=50 examples in original sequential order (no filtering) to simulate a cold-start session.

Hardware. NVIDIA A100 (40GB), Batch Size 1.

Models. TinyLlama-1.1B [7], Phi-3-Mini [2], Mistral-7B [1]. Inference uses greedy decoding (T=0).

Metrics. We define our key metrics as:

- **OMA (Option-Mapped Accuracy):** We embed the generated answer and each option text using `all-MiniLM-L6-v2` and select the option with the highest cosine similarity.
- **SemSim:** Cosine similarity between the SBERT embeddings (`all-MiniLM-L6-v2`) of the generated answer and the gold answer.
- **Latency:** End-to-end wall clock time per query, including retrieval, generation, and reflection overhead.

Note: Latency variations between models may reflect differing inference backends (e.g., optimized vs. eager execution) used by the harness.

4 Results

4.1 The Capacity Sweet Spot

Table 1 presents results from the `results_models` suite. We report the WM-512 configuration with utility scoring for all models.

Table 1: Main Results. TinyACE (WM-512) improves Phi-3 but hurts TinyLlama. All TinyACE runs use utility scoring.

Model	Config	OMA (%)	Sem Sim	Lat (s)
TinyLlama (1.1B)	Baseline	**72.0**	**0.659**	**0.97**
	TinyACE (WM-512)	46.0	0.392	7.85
Phi-3 Mini (3.8B)	Baseline	74.0	0.385	8.52
	TinyACE (WM-512)	**78.0**	0.480	12.0
Mistral (7B)	Baseline	**96.0**	**0.823**	**0.41**
	TinyACE (WM-512)	94.0	0.796	1.81

TinyLlama (1.1B) collapses under the cognitive load. **Phi-3 (3.8B)** sits in the **"Sweet Spot,"** gaining +4% accuracy (37/50 → 39/50 correct answers) despite latency overheads.

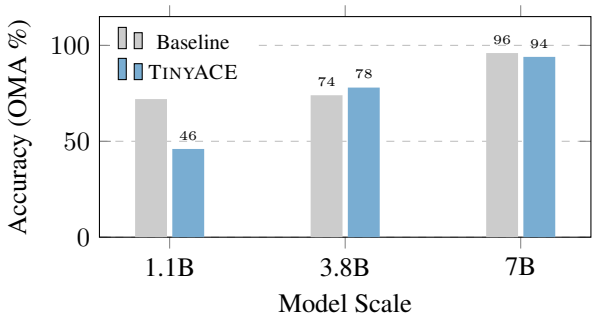

Figure 2: The Capacity Sweet Spot. Only the mid-sized 3.8B model benefits from TinyACE; smaller models collapse, while larger ones saturate.

Mistral (7B) saturates the benchmark (96% baseline). In our SciQ setup, we observe that strong models do not benefit from this form of memory augmentation.

4.2 Ablation Study (Phi-3)

Table 2 uses data from the `results_ablation` suite. Note that baseline latency here (6.43s) differs from Table 1 due to different runner settings.

Table 2: Phi-3 Ablation (WM-256). Data source: `results_ablation`. FIFO performs best among tested variants.

Configuration	OMA	SemSim	Lat
Baseline	74%	0.385	6.43s
TinyACE (FIFO)	**78%**	0.469	8.99s
No Recency	72%	**0.495**	8.83s
No Failure Track	70%	0.466	8.88s
No Vagueness	72%	0.421	9.29s

5 Ablation Study Insights

To understand performance drivers, we analyze the components of Eq. 1 and compare the scoring policy against FIFO eviction (Figure 3).

5.1 Equation Component Analysis

1. Failure Tracking (Most Critical). Removing the failure penalty ($\beta = 0$) yielded the lowest performance (70%), falling 4 points below the baseline (74%). This confirms that explicitly penalizing failures is essential; without it, the model cannot distinguish between helpful and harmful strategies, leading to regression.

2. Recency Bias (The Trade-off). Removing recency bias ($\gamma = 0$) dropped accuracy to 72% but achieved the highest Semantic Similarity (0.495). While recency helps the model "snap" to the correct multiple-choice option, retaining older,

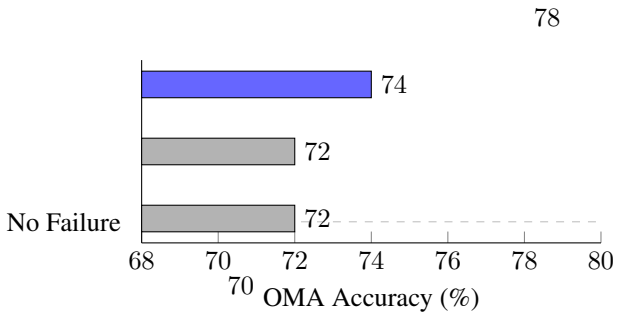

Figure 3: Ablation Impact (Phi-3). Failure tracking is critical; removing it drops performance below baseline.

diverse strategies appears to improve the semantic richness of the explanation.

3. Vagueness Detection (Size Control). Without vagueness filtering ($\delta = 0$), the playbook bloated to its largest size (17 entries) with lower accuracy (72%). This term is critical for efficiency, acting as a gatekeeper against generic entries that consume the token budget without adding reasoning value.

5.2 Policy Comparison

FIFO Eviction (Surprisingly Effective). Simple FIFO eviction achieved the best OMA (78%), matching or exceeding complex scoring. By bypassing Eq. 1 entirely and strictly evicting the oldest entries, we find that for this SLM task, ensuring context "freshness" is often more valuable than complex utility maximization.

6 Future Work

Our findings on the n=50 slice suggest a "Sweet Spot" for mid-sized SLMs, but wider validation is needed. Future work will extend this evaluation to the full SciQ test set and other reasoning benchmarks (e.g., GSM8K) to measure long-term stability and variance. We also plan to investigate **adaptive token budgeting**, where $T_{\max}$ scales dynamically with query complexity, and **vector-based retrieval** to replace the current similarity heuristics, potentially improving recall for non-sequential tasks.

7 Conclusion

TINYACE demonstrates that for mid-range edge-scale models (3B–4B), a bounded working memory effectively bridges reasoning gaps. While latency increases, the accuracy gains (+4%) validate the approach for offline tasks. Crucially, complex memory management appears unnecessary for this scale; a simple FIFO buffer with failure tracking provides the optimal balance.

References

[1] Albert Q. Jiang et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[2] Microsoft Research. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[3] Charles Packer et al. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[4] Noah Shinn et al. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

[5] Guangtao Wang et al. Llms know what to drop: Self-attention guided kv cache eviction. *arXiv preprint arXiv:2503.08879*, 2025.

[6] Johannes Welbl, Nelson F. Liu, and Matt Gardner. Sciq: Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*, 2017.

[7] Peiyuan Zhang et al. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

[8] Qizheng Zhang et al. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.