FROM CONTEXT DISTILLATION TO LATENT MEMORY MANAGEMENT: SELECTING AND ACTIVATING DISTILLED MEMORIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Context distillation internalizes a document into model parameters, allowing subsequent queries to be answered without repeatedly including the document in the prompt. While existing work primarily studies how to construct a memory from a single document, deploying a collection of such memories introduces two additional decisions: which memory is relevant to a query and whether the selected memory should be activated at all. We formulate these decisions as *latent memory management* with memory selection and memory activation, and introduce cache-compatible context distillation, which represents documents as independent LoRA adapters trained to operate on a query prefix KV cache produced by the frozen base model. This shared cache supports latent memory management without recomputing the query prefix, enabling an efficient system in which external retrieval produces a candidate set, internal routing selects one candidate, and Self-Gating decides whether to activate the selected memory. Across NarrativeQA, SQuAD, and LongBench, routing consistently improves over top-1 retrieval, while Self-Gating substantially recovers performance on context-agnostic queries with only small changes to document-specific QA. The same selection strategy also improves latent-token memories, suggesting that the management problem extends beyond LoRA-based representations. Our caching pipeline further reduces memory-construction time by up to 8.4×, peak memory by 4.1×, and training FLOPs by 21.9×, while adding little online management overhead. Overall, our results show that scaling context distillation requires not only writing memories, but also selecting when and which memories to use.

1 INTRODUCTION

Large language models often answer repeated queries about the same documents by conditioning on their text at inference time (Brown et al., 2020; Lewis et al., 2020). This approach is flexible, but processing retrieved text adds computation and memory usage (Tay et al., 2022). Context distillation offers an alternative: it trains a model to reproduce document-conditioned behavior without receiving the document at inference time (Snell et al., 2022; Bhargava et al., 2024; Ye et al., 2026). When a document is reused sufficiently often, the cost of writing it into parameters can be amortized over subsequent queries.

However, distilling one document does not by itself make the resulting memory usable within a collection. At inference time, the relevant memory is generally unknown, and some queries require no document-specific memory at all. A deployed system must therefore determine both *which memory to select* and *whether to activate it*. Oracle memory selection bypasses the first decision. Sequentially accumulating documents in one adapter (Wang et al., 2024; Cao et al., 2025) provides compact storage, but exposes

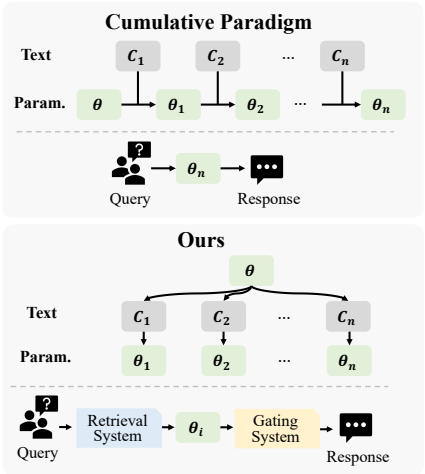


Figure 1: Cumulative writing updates a shared parameter state, whereas our method stores document-specific adapters and manages with selection and activation.

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.

memories to interference and does not provide explicit control over individual memories. We study these decisions as a *latent memory management* problem.

Managing parameterized memories introduces a computational challenge. Memory selection requires evaluating candidates, while rejecting a candidate during memory activation requires returning to the base model. With conventional adapters, prefix representations depend on the active adapter, so switching models can require recomputing the query's KV cache. Simply reusing a base model cache (Wang et al., 2024) with an adapter trained on its own prefix states creates a training–inference mismatch. These costs and compatibility requirements tie memory construction to management.

We address this challenge through *cache-compatible context distillation*. Each document is distilled into an independent LoRA adapter trained to generate from a query-prefix cache computed by the frozen base model. This establishes a shared prefix cache for adapter evaluation and base-model fallback, while teacher-side prefix reuse further reduces the repeated computation required to construct latent memories. Building on this shared cache, memory selection proceeds in two stages: external retrieval first identifies a set of candidate memories, and internal routing then selects among them using both retrieval-based and model-derived signals. Finally, Self-Gating controls memory activation by deciding whether to apply the selected adapter or fall back to the base model.

Our main evaluation examines distillation, memory selection, and memory activation on NarrativeQA and SQuAD across Qwen and Llama backbones. Our proposed method achieves the overall strongest performance, while largely preserving the base model's general capabilities. Additional experiments on LongBench demonstrate that the proposed memory management approach generalizes to substantially longer contexts, improving macro QA F1 by up to 9.6 points across different backbones. Applying the memory management to latent-token memories also improves downstream QA, suggesting that the approach is not limited to LoRA-based latent memories. The efficiency experiments further show that cache-compatible distillation reduces memory construction time by up to $8.4\times$, peak memory by $4.1\times$, and training FLOPs by $21.9\times$, while introducing only limited internal-routing and Self-Gating overhead at inference time.

Our contributions are threefold:

- We formulate latent memory management with memory selection under unknown query–document correspondence, and memory activation for mixed context-specific and context-agnostic queries.
- We introduce cache-compatible context distillation, which stores documents as independent adapters sharing a query prefix cache and reuses the same candidate probe for routing and fallback decisions.
- We separately quantify the gains from routing, the benefits and cost of Self-Gating, and the importance of shared-prefix consistency between training and inference.

2 Background and Related Work

Context distillation. Given a document c and query q, context distillation trains a parameter update $\Delta\theta$ to match a document-conditioned teacher with a context-free student:

$$D_{\mathrm{KL}}(\pi_\theta(\cdot \mid c, q) \parallel \pi_{\theta+\Delta\theta}(\cdot \mid q)) . \tag{1}$$

This idea underlies context internalization (Snell et al., 2022) and prompt baking (Bhargava et al., 2024). On-Policy Context Distillation (Ye et al., 2026) and OPSDL (Zhang et al., 2026) modify the response distribution used for distillation. Text-to-LoRA (Charakorn et al., 2025), Doc-to-LoRA (Charakorn et al., 2026), and SHINE (Liu et al., 2026) use hypernetworks to produce parameter updates from textual inputs. Plug-and-play knowledge modules (Caccia et al., 2025) further connect distilled knowledge with retrieval. These writing mechanisms focus on how to write modular memories, while our focus is to manage the latent memories with the cache-compatible distillation.

Context Compression. Compression can retain a shorter text prompt, as in LLMLingua (Jiang et al., 2023) and LLMLingua-2 (Pan et al., 2024), or provide compact learned representations, as in xRAG (Cheng et al., 2024) and PISCO (Louis et al., 2025). Cartridges (Eyuboglu et al., 2025), context cascade compression (Liu & Qiu, 2025), and latent context compilation (Li et al., 2026b) study additional compact context representations. LCLM (Li et al., 2026a) learns encoder–decoder compression at scale. While these approaches focus primarily on compressing context into shorter or more compact representations, our work internalizes documents as parameterized memories and develops mechanisms for their selection and activation.

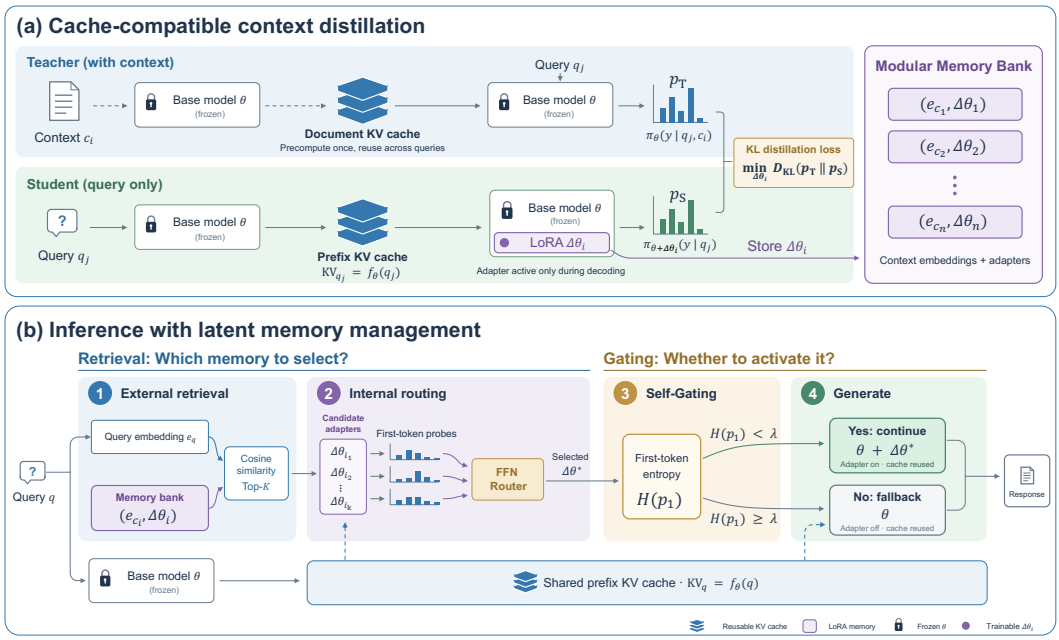


Figure 2: Overview of memory construction and management. Each document is distilled into an independent adapter. Teacher-side cache reuse avoids repeated document encoding during distillation. Cache-compatible student training enables candidate adapters and base model fallback to share the query prefix cache. Routing determines which memory to select, and Self-Gating determines whether to activate it.

Cumulative writing. TempLoRA (Wang et al., 2024) and InfiniteICL (Cao et al., 2025) cumulatively update a shared parameter state as documents arrive. This keeps adapter storage fixed, whereas a bank of independent adapters grows with the number of documents. Such sequential updates are susceptible to catastrophic forgetting, as studied by Learning without Forgetting (Li & Hoiem, 2017) and elastic weight consolidation (Kirkpatrick et al., 2017). Under a recursive distillation assumption, cumulative writing can also be interpreted as reproducing conditioning on accumulated contexts (Appendix C), which implies the limitation of this paradigm.

3 Methodology

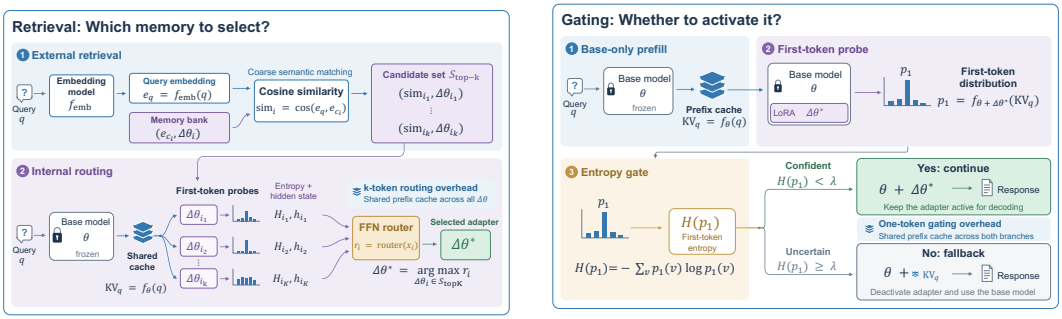
We formulate context distillation as the construction and management of independent latent memories. Given a collection of documents $\mathcal{C} = \{c_1, \ldots, c_M\}$, we freeze the base model f_θ and distill each document c_i into a separate LoRA adapter $\Delta\theta_i$. The resulting memory bank is $\mathcal{M} = \{(\mathbf{e}_{c_i}, \Delta\theta_i)\}_{i=1}^{M}$, where $\mathbf{e}_{c_i}$ supports external retrieval. At inference time, memory management addresses two questions: *which memory to select* and *whether to activate it*.

As illustrated in Figure 2, our key design is **cache-compatible context distillation**: adapters are trained to operate on query-prefix KV states produced by the frozen base model. This training scheme makes all adapters compatible with the same frozen-base query prefix, enabling cache sharing during candidate evaluation and base-model fallback. We first describe cache-compatible memory construction, then introduce external retrieval and internal routing for memory selection, followed by Self-Gating for memory activation.

3.1 Cache-Compatible Context Distillation

For each document c_i, we generate synthetic queries $\{q_j\}_{j=1}^{n_i}$ and obtain teacher responses y_j from the base model conditioned on c_i. The student receives the query without the document and learns to reproduce the teacher's answer-token distributions through $\Delta\theta_i$. Each adapter is trained independently, so constructing a new memory does not overwrite previously stored adapters.

Cache-compatible student. Ordinary adapter inference produces adapter-dependent query-prefix states, preventing direct reuse of one prefix cache across different candidates. We instead compute the query-prefix KV cache KV_q with the frozen base model and train each adapter to predict answers conditioned on this shared cache. The same computation is used during training and deployment, making the resulting adapters compatible with a common base-model prefix.



(a) Which memory to select.

(b) Whether to activate it.

Figure 3: Memory management enabled by cache-compatible distillation. Routing determines which memory to select, and Self-Gating determines whether to activate it.

Denote the cache-compatible student distribution by $\pi^{\mathrm{CC}}_{\theta,\Delta\theta_i}$. We optimize the forward-KL objective

$$\mathcal{L}_i = \frac{1}{Z_i}\sum_{j=1}^{n_i}\sum_{t=1}^{|y_j|} D_{\mathrm{KL}}\Big(\pi_\theta(\cdot \mid c_i, q_j, y_{j,<t}) \;\Big\|\; \pi^{\mathrm{CC}}_{\theta,\Delta\theta_i}(\cdot \mid KV_{q_j}, y_{j,<t})\Big), \tag{2}$$

where $Z_i = \sum_j |y_j|$ normalizes over supervised answer tokens.

Teacher-side cache reuse. For a fixed document, the teacher repeatedly processes the same document-containing prefix across synthetic queries. We precompute the KV cache of this common prefix and reuse it during teacher response generation and scoring, while processing query-dependent tokens separately. This avoids repeated document encoding without changing the teacher's conditioning information.

Cache-compatible distillation for management. Because every student is trained to operate on frozen-base prefix states, different adapters can be evaluated from the same query-prefix cache KV_q without prefilling the query again. We refer to this property as ***hot-swapping***. Each candidate maintains its own adapter-dependent suffix states, while the shared prefix remains unchanged. Hot-swapping therefore removes repeated prefix computation while retaining candidate-specific forward passes. This cache-compatible mechanism supports both internal routing and base model fallback, and the detailed cache boundaries are specified in Appendix A.5.

3.2 Which Memory to Select: Retrieval and Routing

Independent adapters turn context access into a memory selection problem. Figure 3 summarizes the selection and activation pipeline. For selection, we use the two-stage system shown in Figure 3a: *external retrieval* narrows the memory bank to a small candidate set, and *internal routing* selects a candidate using signals from the language model itself.

External retrieval. An embedding model f_{emb} encodes each document and the incoming query:

$$\mathbf{e}_{c_i} = f_{\mathrm{emb}}(c_i), \qquad \mathbf{e}_q = f_{\mathrm{emb}}(q). \tag{3}$$

We retrieve the k adapters whose document embeddings have the highest cosine similarity:

$$a_i = \cos(\mathbf{e}_q, \mathbf{e}_{c_i}), \qquad \mathcal{S}_k(q) = \mathrm{TopK}_{i\in\{1,\dots,M\}}\, a_i. \tag{4}$$

Internal routing. Embedding similarity measures semantic relevance, but does not directly capture how the language model responds when a candidate memory is activated. We therefore complement retrieval scores with adapter-conditioned predictive features.

Using the cache-compatible adapters, we first compute the shared query-prefix cache KV_q once. For each $i \in \mathcal{S}_k(q)$, we briefly activate $\Delta\theta_i$ to obtain its first-token distribution:

$$p_{i,1} = \pi^{\mathrm{CC}}_{\theta,\Delta\theta_i}(\cdot \mid KV_q). \tag{5}$$

This probe provides adapter-conditioned predictive signals without generating a complete candidate answer. A lightweight multilayer perceptron combines the retrieval and probe signals to score each candidate:

$$i^* = \arg\max_{i \in \mathcal{S}_k(q)} r_\phi(x_i), \tag{6}$$

where x_i denotes the candidate's combined retrieval and model-derived features. The exact feature construction and its ablation are provided in Appendix B.1. Because all candidate probes reuse KV_q, routing requires one shared query prefill followed by k first-token probes, rather than k complete query prefill passes.

3.3 Whether to Activate It: Self-Gating

Retrieval and routing always return a candidate, even when the query does not require any stored document. Selecting the best available memory therefore does not establish that it should be activated. Given the selected memory, Self-Gating determines whether to activate it before answer generation, as illustrated in Figure 3b.

Why first-token entropy? Figure 4 shows that context-specific queries have lower first-token entropy than context-agnostic queries under both ICL and distilled LoRA adapters. In contrast, their base-model entropy distributions substantially overlap, indicating that dataset identity or query format alone provides only a weak separation signal. The similar pattern under ICL and LoRA therefore motivates using the adapter's predictive confidence as an activation signal. The consistent separation under both ICL and distilled LoRA motivates using adapter-conditioned entropy as a lightweight activation score. We quantify the base-only signal and provide experimental analysis in Appendix B.2.

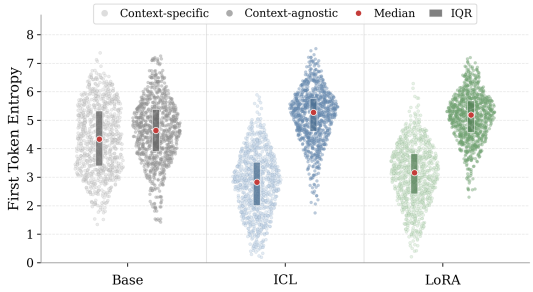


Figure 4: First-token predictive entropy.

Activation and fallback. For the selected adapter $\Delta\theta_{i^*}$, we compute

$$H_{i^*} = -\sum_{v \in \mathcal{V}} p_{i^*,1}(v) \log p_{i^*,1}(v), \qquad g(q) = \mathbf{1}[H_{i^*} < \lambda], \tag{7}$$

where λ is chosen on a calibration split and frozen before evaluation. The gate has two branches:

- **Continue ($H_{i^*} < \lambda$).** Retain the selected adapter and generate the answer using its first-token distribution and candidate-specific suffix cache.
- **Fallback ($H_{i^*} \geq \lambda$).** Discard the adapter-dependent suffix and reuse KV_q to obtain the base model's first-token distribution, then generate without an adapter.

Reusing the routing computation. When internal routing has already probed the selected adapter, Self-Gating reuses its first-token distribution and entropy. For top-1 retrieval without an internal router, the gate obtains these quantities through a single first-token probe. Fallback requires a base model first-token computation, but does not repeat the query prefix prefill. Cache-compatible distillation thus enables both candidate selection and activation through a common shared prefix cache.

4 Experiments

We evaluate Qwen2.5-0.5B and Qwen2.5-7B (Yang et al., 2024), together with Llama-3.2-1B (Grattafiori et al., 2024), on NarrativeQA (Kočiskỳ et al., 2018) and SQuAD (Rajpurkar et al., 2016) for context-specific QA, CommonsenseQA (Talmor et al., 2019) for context-agnostic QA, and three LongBench tasks (Bai et al., 2024) for long-context evaluation. We use Qwen3-Embedding-0.6B (Zhang et al., 2025) to compute similarity. Full details are provided in Appendix A.

We evaluate the proposed latent memory management through four questions:

- Which memory should be selected for a query?

- Whether the selected memory should be activated?
- Whether cache compatibility makes construction and management efficient?
- Whether the conclusions extend to long-context QA and other latent memories?

Table 1: Main results on NarrativeQA and SQuAD across Qwen2.5-0.5B, Qwen2.5-7B, and Llama-3.2-1B. Best results excluding the in-context upper bound are shown in bold. † denotes access to the ground-truth context.

Method	NarrativeQA						SQuAD					
	Qwen2.5-0.5B		Qwen2.5-7B		Llama-3.2-1B		Qwen2.5-0.5B		Qwen2.5-7B		Llama-3.2-1B	
	R-1 ↑	R-L ↑	R-1 ↑	R-L ↑	R-1 ↑	R-L ↑	EM ↑	F1 ↑	EM ↑	F1 ↑	EM ↑	F1 ↑
Lower Bound												
Base Model	13.04	12.21	14.81	14.25	9.20	8.69	2.27	9.46	13.25	21.74	4.16	9.93
Upper Bound												
In-context†	40.57	39.53	55.05	54.09	42.12	41.08	41.86	57.51	77.55	84.88	34.14	49.17
Context Compression												
LLMLingua@1	14.44	13.58	18.21	17.76	11.23	10.57	22.22	34.59	46.06	53.48	4.01	14.60
LLMLingua@3	8.99	8.20	13.39	12.97	10.13	9.48	7.00	15.68	26.53	33.09	1.85	9.50
LLMLingua@5	8.00	7.29	11.80	11.38	9.70	9.09	5.64	14.22	19.87	26.58	1.17	7.99
LLMLingua-2@1	16.15	15.65	16.03	15.73	8.72	8.20	11.51	20.58	20.25	29.15	0.08	5.46
LLMLingua-2@3	14.38	13.83	12.97	12.64	8.47	8.04	9.99	19.00	17.34	25.41	0.04	5.10
LLMLingua-2@5	13.37	12.83	12.64	12.30	8.68	8.24	7.91	16.15	16.28	24.12	0.08	5.18
xRAG@1	13.70	12.86	12.45	11.65	8.63	8.13	5.15	17.12	14.84	23.28	0.64	6.59
PISCO@1	17.00	16.12	22.70	22.16	11.43	10.79	14.46	34.86	**55.34**	**63.29**	4.09	15.84
Cumulative Paradigm												
TempLoRA	4.67	4.18	4.83	4.11	4.41	4.04	0.00	1.14	0.00	1.60	0.00	1.19
InfiniteICL	10.10	9.35	4.24	3.77	5.20	4.89	0.76	1.25	0.91	0.91	0.00	1.34
TempLoRA-CD	6.65	6.14	4.09	3.78	7.16	6.78	0.00	0.64	0.00	0.00	0.04	1.69
Ours												
Ours@1	23.47	22.54	30.12	29.26	22.85	22.10	26.73	41.26	34.85	44.94	23.35	36.27
Ours@3	24.47	**23.95**	32.21	31.32	23.39	22.62	**28.30**	43.11	36.34	46.57	24.16	37.47
Ours@5	**24.54**	23.94	**32.61**	**31.72**	**23.44**	**22.66**	28.14	**43.25**	36.30	46.45	**24.54**	**37.74**

4.1 Which Memory to Select?

Table 1 evaluates memory selection when the gold query–document correspondence is unavailable. We compare three approaches: compressing retrieved context into shorter text or latent representations, storing all documents in a cumulatively updated parameter state, and retrieving from independently stored document adapters. Under realistic setting evaluation, the cumulative baselines score below the independently addressable adapters, consistent with interference across sequential updates. We provide a more detailed analysis of the cumulative paradigm in Appendix C.

Across both datasets and all three backbones, our@1, ours@3, and ours@5 variants achieve state-of-the-art performance in most of the tasks, demonstrating the effectiveness of independently stored and selectively accessed memories across different model scales and tasks. Expanding the candidate set consistently improves performance over top-1 retrieval: both Ours@3 and Ours@5 outperform Ours@1 on every reported metric. For example, on SQuAD, Ours@3 increases F1 from 41.26 to 43.11, from 44.94 to 46.57, and from 36.27 to 37.47 for Qwen2.5-0.5B, Qwen2.5-7B, and Llama-3.2-1B, respectively, while Ours@5 achieves further strong results across all three backbones. The consistent gains from expanding the candidate set show that access to a broader set of memories increases the opportunity to recover task-relevant information, and that our selection mechanism can effectively identify and exploit the most useful memory. These results establish memory selection as a critical component of modular latent memory management.

4.2 Whether to Activate the Selected Memory?

A retrieved memory should not always be activated, since an irrelevant memory may interfere with queries that the base model can answer directly. To evaluate this decision, we construct a hybrid

Table 2: Hybrid-query results on NarrativeQA and CommonsenseQA. Best and second-best results in each column are shown in **bold** and underlined, respectively.

Setting	NarrativeQA						CommonsenseQA		
	Qwen2.5-0.5B		Qwen2.5-7B		Llama-3.2-1B		Qwen2.5-0.5B	Qwen2.5-7B	Llama-3.2-1B
	R-1 ↑	R-L ↑	R-1 ↑	R-L ↑	R-1 ↑	R-L ↑	Acc. ↑	Acc. ↑	Acc. ↑
Base Model	13.04	12.21	14.81	14.25	9.20	8.69	**48.48%**	**85.50%**	**39.31%**
Context Compression									
xRAG@1	13.70	12.86	12.45	11.65	8.63	8.13	39.23%	83.05%	37.10%
PISCO@1	17.00	16.12	22.70	22.16	11.43	10.79	31.70%	76.25%	29.48%
Context Distillation									
TempLoRA	4.67	4.18	4.83	4.11	4.41	4.04	20.56%	20.31%	18.18%
InfiniteICL	10.10	9.35	4.24	3.77	5.20	4.89	19.49%	20.56%	20.23%
TempLoRA-CD	6.65	6.14	4.09	3.78	7.16	6.78	19.57%	18.92%	19.41%
Ours									
Gating@1	22.74	21.84	30.15	29.28	21.85	21.11	45.62%	84.93%	38.74%
Gating@3	23.40	22.43	32.22	31.33	**22.18**	**21.42**	47.91%	85.18%	38.25%
Gating@5	**23.44**	**22.48**	**32.64**	**31.74**	**22.18**	21.41	47.34%	85.26%	38.66%

setting that mixes context-specific NarrativeQA queries with context-agnostic CommonsenseQA queries. For each query, the gating mechanism decides whether to activate the selected latent memory or fall back to the base model without memory.

As shown in Table 2, across all three backbones, the gated variants retain strong context-specific QA performance while keeping the best context-agnostic tasks accuracy within 0.57 percentage points of the corresponding base model. On NarrativeQA, it obtains 23.44, 32.64, and 22.18 ROUGE-1 on Qwen2.5-0.5B, Qwen2.5-7B, and Llama-3.2-1B, respectively. Compared with the strongest context-compression baseline for each backbone, these results represent improvements of 6.44, 9.94, and 10.75 points. Gating@5 also substantially outperforms all context distillation baselines across the three backbones, demonstrating the advantage of independently stored and selectively activated memories over compressed contexts and cumulatively updated parameter states.

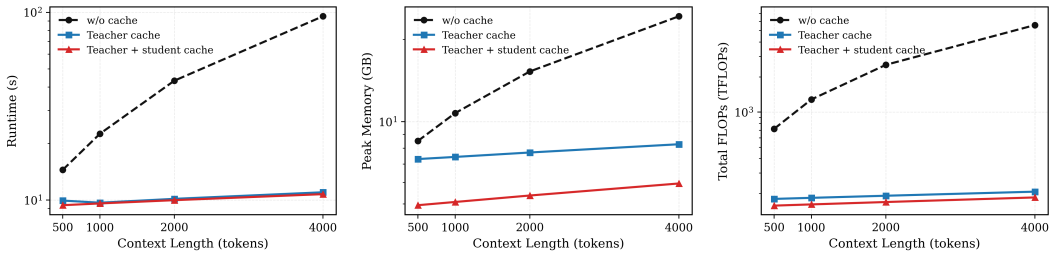
Expanding the candidate set generally provides further gains. From Gating@1 to Gating@5, ROUGE-1 increases from 22.74 to 23.44 on Qwen2.5-0.5B, from 30.15 to 32.64 on Qwen2.5-7B, and from 21.85 to 22.18 on Llama-3.2-1B. Meanwhile, the gated setting for each backbone remains most of capability on context-agnostic QA. The best gated settings achieve 47.91%, 85.26%, and 38.74% accuracy across the three backbones, compared with 48.48%, 85.50%, and 39.31% for the corresponding base models. Thus, selective activation enables the model to exploit context-specific memories while largely retaining its general capabilities.

4.3 Efficiency from Cache Compatibility

Cache compatibility affects both stages of the system. During memory construction, the teacher reuses the document-prefix cache across distillation queries and the student reuses the frozen-base query-prefix cache. During inference, candidate adapters share that same query prefix, so routing and gating add short adapter-specific probes rather than repeating a complete prefill. Full measurement details are in Appendix D.

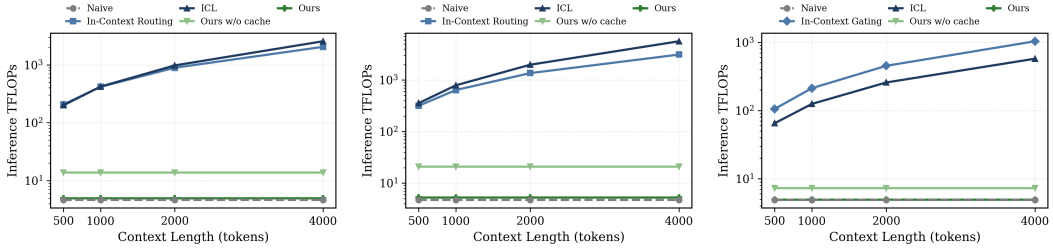
Memory construction. Figure 5 compares no caching, teacher-side caching, and joint teacher–student caching. At 4,000 context tokens, the full cache-compatible pipeline reduces training time by $8.4\times$, peak memory by $4.1\times$, and training FLOPs by $21.9\times$ relative to the uncached implementation. Teacher reuse supplies most of the runtime saving, while student reuse further reduces computation and, crucially, produces adapters that can share a frozen-base prefix at inference time.

Memory selection and activation. Figure 6 separates document internalization from cache reuse. At 4,000 tokens, our routing pipeline uses $414.7\times$ and $1081.4\times$ fewer model FLOPs than text-conditioned in-context routing with three and five candidates, respectively. These end-to-end ratios include the benefit of removing document text from the prompt. The comparison with Ours w/o Cache



(a) Training time (b) Peak memory (c) Training FLOPs

Figure 5: Memory construction efficiency. Teacher-prefix reuse avoids repeatedly encoding the document, while student-prefix reuse additionally avoids recomputing the shared query prefix.



(a) Routing with three candidates (b) Routing with five candidates (c) Selective activation

Figure 6: Latent memory management efficiency. ICL processes retrieved text; in-context routing/gating probes text-conditioned models; Ours w/o Cache recomputes adapter prefixes. Cache-compatible management reuses base model prefix cache across candidate adapters.

isolates prefix sharing itself, which provides additional $2.8\times$ and $4.0\times$ reductions. For activation, Self-Gating adds approximately 1% over unmanaged adapter generation and uses $210.0\times$ fewer FLOPs than in-context gating in this fixed-output benchmark.

Amortization. Latent memory incurs a one-time construction cost but reduces the cost of repeatedly querying the same document. Let C_{build} denote the cost of constructing a memory, and let C_{text} and C_{memory} denote the per-query costs of textual retrieval and latent-memory inference. The number of queries required to amortize memory construction is

$$N_{\text{BE}} = \left\lceil \frac{C_{\text{build}}}{C_{\text{text}} - C_{\text{memory}}} \right\rceil, \qquad C_{\text{text}} > C_{\text{memory}}. \tag{8}$$

Figure 7 shows that the break-even point decreases as context length increases. Longer contexts make repeated textual inference more expensive, whereas latent-memory inference avoids re-encoding the document for every query. Cache compatibility further lowers the break-even point by reducing both memory-construction cost and the overhead of routing and activation. In contrast, without cache compatibility, the construction cost remains high and requires substantially more document reuse to amortize. These results suggest that latent memory is most beneficial for long documents that are queried repeatedly, while textual RAG remains preferable when documents receive only a small number of queries.

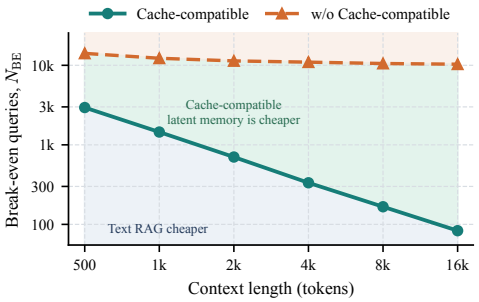


Figure 7: Break-even number of queries under different context lengths. Below each curve, textual RAG is cheaper, and above it, latent-memory inference is cheaper.

4.4 Extensions of Latent Memory Management

We evaluate whether our memory management approach extends along two dimensions: from short to long contexts, and from LoRA adapters to latent tokens.

Extension to long contexts. We use three QA tasks from LongBench(2WikiMQA, MultiFieldQA-en, and Qasper) containing 460 contexts and 550 questions in total. We report macro F1 by averaging

Table 3: Experiments on LongBench. Avg. tokens denotes the average number of additional memory tokens.

Backbone	Memory	Avg. tokens	No context	Ours@1	Ours@3	Ours@5
Qwen2.5-0.5B-Instruct	LoRA adapter	0	17.04	19.74	19.80	**19.97**
Llama-3.2-1B-Instruct	LoRA adapter	0	17.20	25.93	26.72	**26.81**
Qwen2.5-7B-Instruct	LoRA adapter	0	17.84	24.14	24.54	**25.20**
LCLM	Latent tokens	406	18.67	31.31	32.24	**32.35**

the three dataset-level scores. As shown in Table 3, the retrieved LoRA memories outperform the no-context model across all three backbones. Ours@5 improves macro F1 from 17.04 to 19.97 on Qwen2.5-0.5B, from 17.20 to 26.81 on Llama-3.2-1B, and from 17.84 to 25.20 on Qwen2.5-7B. Increasing the candidate set from one to five also consistently improves performance, indicating that the proposed selection mechanism remains effective for substantially longer documents.

Extensions to other latent memories. We further apply the same selection mechanism to LCLM, which represents documents using compressed latent tokens rather than LoRA adapters. Routing improves its macro F1 from 31.31 with top-1 retrieval to 32.24 with three candidates and 32.35 with five candidates. This consistent improvement suggests that our selection mechanism is not specific to cache-compatible LoRA memories, but can also benefit other modular latent-memory representations. Because LCLM uses a different backbone and training protocol, its absolute scores are not directly comparable with the LoRA results.

4.5 Ablation Study

Ablation on Routing. Both retrieval and first-token features contribute to memory selection. Retrieval features provide the primary signal by measuring the semantic alignment between a query and each candidate memory, and removing them leads to a larger performance drop. First-token features provide complementary model-derived evidence, allowing the router to refine the ranking beyond retrieval similarity alone. Together, the two feature sources yield more reliable selection(Appendix B.1).

Ablation on Self-Gating. We first examine whether the gating decision exploits dataset-specific differences in query distributions. A gate based only on base-model entropy achieves 56.84% balanced accuracy, substantially below the 79.00% achieved by Self-Gating, indicating that dataset-level query characteristics alone do not account for its performance. Self-Gating is also insensitive to the gating threshold, maintaining stable performance across a broad threshold range. By selectively activating memories based on adapter-conditioned entropy, Self-Gating retains most context-specific gains while recovering general knowledge performance (Appendix B.2).

Ablation on Cache-Compatible Distillation. Cache-compatible distillation achieves performance comparable to conventional distillation across four representative objectives, indicating that the shared-prefix construction preserves memory quality while enabling cache reuse. The comparison between training and inference configurations further highlights the importance of consistency: adapters trained with the shared-prefix formulation maintain context-specific performance and retain substantially more general capability under shared-prefix inference(Appendix B.3).

5 Conclusion

We formulate the use of context-specific parameter memories as a selection and activation problem. Cache-compatible distillation trains adapters to use a frozen base prefix cache for candidate evaluation and base-model fallback. On NarrativeQA and SQuAD, retrieval and routing improve document-specific question answering, while Self-Gating preserves most of the base model's performance on context-agnostic queries. The long-context experiments show consistent gains across three backbones, with improvements of up to 9.6 macro F1 points over no-context inference. Routing also improves downstream QA for latent-token memories, showing that the proposed memory management approach is not limited to LoRA adapters. Our experiment shows that cache-compatible distillation reduces construction time by up to $8.4\times$, peak memory by $4.1\times$, and training FLOPs by $21.9\times$, while keeping the additional routing and gating cost low. These results show that context distillation requires not only constructing document memories, but also selecting and activating them efficiently.

AI Disclosure

In this work, we used generative AI tools to generate synthetic data, polish writing and assist with partial implementation of our methods. Specifically, we used Qwen2.5-7B to generate synthetic questions for latent memory construction, as described in Appendix A.3. All generated questions were filtered using the predefined criteria described in the paper. Generative AI tools also assisted with implementing part of the experimental code. Additionally, we used generative AI tools for manuscript editing and language refinement.

We have not used generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, translate text, or clean and reformat datasets. Providing critical ingredients for mathematical proofs, writing proofs, and supporting qualitative or thematic data analysis are not applicable to this work.

We have reviewed all AI-assisted work, including the synthetic data generation, AI-assisted code, and manuscript edits. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

Reproducibility Statement

Appendix A specifies the benchmark construction, training and evaluation settings. Appendix B provides the router, Self-Gating, and cache-compatible distillation ablations, while Appendix D describes the efficiency measurement setting. Code will be released with the final paper.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. In *Proceedings of ACL*, pp. 3119–3137, 2024. URL https://aclanthology.org/2024.acl-long.172/.

Aman Bhargava, Cameron Witkowski, Alexander Detkov, and Matt Thomson. Prompt baking. *arXiv preprint arXiv:2409.13697*, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Lucas Caccia, Alan Ansell, Edoardo Ponti, Ivan Vulić, and Alessandro Sordoni. Training plug-n-play knowledge modules with deep context distillation. *arXiv preprint arXiv:2503.08727*, 2025.

Bowen Cao, Deng Cai, and Wai Lam. Infiniteicl: Breaking the limit of context window size via long short-term memory transformation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11402–11415, 2025.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-lora: Instant transformer adaption. *arXiv preprint arXiv:2506.06105*, 2025.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-lora: Learning to instantly internalize contexts. *arXiv preprint arXiv:2602.15902*, 2026.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xRAG: Extreme Context Compression for Retrieval-augmented Generation with One Token. *arXiv preprint arXiv:2405.13792*, 2024. URL https://arxiv.org/abs/2405.13792.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, et al. Cartridges: Lightweight and general-purpose long context representations via self-study. *arXiv preprint arXiv:2506.06266*, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In *Proceedings of EMNLP*, pp. 13358–13376, 2023. URL https://aclanthology.org/2023.emnlp-main.825/.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114 (13):3521–3526, 2017.

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

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

Ang Li, Sean McLeish, Haozhe Chen, Nimit Kalra, Zaiqian Chen, Artem Gazizov, Venkata Anoop Suhas Kumar Morisetty, Bhavya Kailkhura, Harshitha Menon, Zhuang Liu, Brian R. Bartoldson, Tom Goldstein, Sanae Lotfi, Micah Goldblum, and Pavel Izmailov. End-to-End Context Compression at Scale. *arXiv preprint arXiv:2606.09659*, 2026a. URL https://arxiv.org/abs/2606.09659.

Zeju Li, Yizhou Zhou, and Qiang Xu. Latent context compilation: Distilling long context into compact portable memory. *arXiv preprint arXiv:2602.21221*, 2026b.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Fanfan Liu and Haibo Qiu. Context cascade compression: Exploring the upper limits of text compression. *arXiv preprint arXiv:2511.15244*, 2025.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. Shine: A scalable in-context hypernetwork for mapping context to lora in a single pass. *arXiv preprint arXiv:2602.06358*, 2026.

Maxime Louis, Hervé Déjean, and Stéphane Clinchant. PISCO: Pretty Simple Compression for Retrieval-Augmented Generation. In *Findings of ACL*, pp. 15506–15521, 2025. URL https://aclanthology.org/2025.findings-acl.800/.

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. LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression. In *Findings of ACL*, pp. 963–981, 2024. URL https://aclanthology.org/2024.findings-acl.57/.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 2383–2392, 2016.

Mrigank Raman, Pranav Mani, Davis Liang, and Zachary Lipton. For distillation, tokens are not all you need. In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. *arXiv preprint arXiv:2601.19897*, 2026.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, 2019.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.

Yan Wang, Dongyang Ma, and Deng Cai. With greater text comes greater necessity: Inference-time training helps long text generation. *arXiv preprint arXiv:2401.11504*, 2024.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv e-prints*, pp. arXiv–2412, 2024.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models. *arXiv preprint arXiv:2602.12275*, 2026.

Xinsen Zhang, Zhenkai Ding, Tianjun Pan, Run Yang, Chun Kang, Xue Xiong, and Jingnan Gu. Opsdl: On-policy self-distillation for long-context language models. *arXiv preprint arXiv:2604.17535*, 2026.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

A Implementation and Evaluation Details

This section describes the benchmarks, model backbones, synthetic questions, and adapter-training settings used in our experiments.

A.1 Benchmarks and metrics

We evaluate document-dependent question answering on NarrativeQA (Kočiskỳ et al., 2018), SQuAD (Rajpurkar et al., 2016), and three LongBench tasks (Bai et al., 2024): 2WikiMQA, MultiFieldQA-en, and Qasper. We use NarrativeQA test summaries, the 300 SQuAD validation contexts with the most associated questions, and 460 LongBench contexts. CommonsenseQA (Talmor et al., 2019) provides context-independent questions for evaluating whether the system can avoid unnecessary memory activation.

Table 4 summarizes the evaluation sets. NarrativeQA is evaluated with ROUGE-1 and ROUGE-L, SQuAD with exact match and F1, CommonsenseQA with accuracy, and LongBench with the macro-average F1 across its three tasks. For our LoRA-based method, evaluation documents are used during memory construction and are not included in the answer generation prompt. Explicit context baselines access the corresponding context at inference time.

Table 4: Evaluation datasets, context-length statistics, and metrics. NarrativeQA uses test summaries, and SQuAD uses the 300 most frequently queried validation contexts.

Dataset	Contexts	Questions	Avg. tokens	Max tokens	Metric
NarrativeQA	355	10,557	731.59	1,349	ROUGE-1 / ROUGE-L
SQuAD	300	2,642	158.93	620	EM / F1
CommonsenseQA	—	1,221	—	—	Accuracy
LongBench	460	550	6,720.67	21,879	Macro F1

A.2 Backbones

We use Qwen2.5-0.5B, Qwen2.5-7B (Yang et al., 2024), and Llama-3.2-1B (Grattafiori et al., 2024). The NarrativeQA, SQuAD, and CommonsenseQA experiments use the base checkpoints, whereas the LongBench experiments use the corresponding Instruct checkpoints. Instruct models use their native chat templates and task instructions; base and Instruct results are reported separately and are not treated as interchangeable checkpoints.

Prompt 1: Query Generation

Read the following story context and generate `{batch_size}` distinct questions based on it.

Requirements:

1. The questions must be highly related to the context.
2. Output the questions as a numbered list, e.g., "1. Question ... 2. Question ...".
3. Do not provide answers, options, or any other text. Only the questions.

Context:

`{context}`

Questions:

A.3 Synthetic question generation

For every document, we use Qwen2.5-7B as the query generator and collect 1,000 accepted questions. The same question strings are reused across the three adapter backbones, while each backbone generates its own teacher targets. For LongBench, this produces 460,000 synthetic questions.

Long documents are divided into 1,800-token windows with 180-token overlap under the generator tokenizer.

The generator uses the Prompt 1 without a chat template. Each call requests 24 questions using temperature 0.9 and top-p 0.95. We retain questions containing 5–45 words and ending in a question mark, remove within-document duplicates, and continue generation until 1,000 questions are accepted for each document. We additionally exclude candidates that exactly match an official evaluation question after normalization or have token-set Jaccard similarity of at least 0.85. Official answers are never used during question generation. The matched teacher backbone then generates a target from the document and synthetic question using greedy decoding with at most 50 new tokens.

A.4 Training hyperparameters

All document adapters use the settings in Table 5. We use the same training hyperparameters across all methods and ablations, including cumulative training, naive distillation, and cache-compatible distillation, to ensure a controlled comparison. The teacher receives the document and question, whereas the student receives the question without the document.

Table 5: Training hyperparameters for LoRA distillation.

Hyperparameter	**Value**
LoRA Configuration	
Rank r	64
Scaling factor α	128
Dropout	0.05
Target modules	Query, key, value, output, gate, up, and down projections
Optimization	
Learning rate	5×10^{-4}
Training epochs	10
Training batch size	16
Learning rate scheduler	Cosine
Generation	
Max new tokens	50

A.5 Cache Boundaries and Reuse Semantics

Shared prefix boundary. Let the tokenized query prompt be $q = (q_1, \ldots, q_L)$, where the logits at the final prompt token q_L define the distribution of the first answer token. We place the shared cache boundary immediately before q_L. With all adapters disabled, the frozen base model processes $q_{<L} = (q_1, \ldots, q_{L-1})$ once and produces

$$KV_q = \mathrm{Cache}_\theta(q_{<L}). \tag{9}$$

The final prompt token is excluded from the shared prefix so that it can be processed under each candidate adapter. Consequently, KV_q contains only states produced by the frozen base model and is independent of the selected memory.

Candidate branches. For each retrieved candidate i, we activate $\Delta\theta_i$ and process q_L from an independent copy of KV_q:

$$\big(p_{i,1}, KV_i^{\mathrm{suf}}\big) = f_{\theta+\Delta\theta_i}\big(q_L \mid \mathrm{copy}(KV_q)\big), \tag{10}$$

where $p_{i,1}$ is the first answer token distribution used by internal routing and Self-Gating, and KV_i^{suf} is the adapter dependent suffix cache beginning at q_L. The shared cache remains unchanged throughout candidate evaluation. Each candidate receives an independent copy and therefore cannot observe states produced by another adapter.

Cache ownership. Only states produced entirely by the frozen base model are shared across candidates. Once a token has been processed with an active adapter, its key and value states and all subsequent decoding states belong to that candidate branch. These states cannot be shared with another adapter or with the base model. The cache boundary therefore separates the common frozen base prefix KV_q from the adapter dependent suffix KV_i^{suf}.

Activation and fallback. If candidate i^* is activated, answer generation continues with $\Delta\theta_{i^*}$ and its candidate specific suffix cache. If Self-Gating chooses fallback, all adapter dependent suffix states are discarded. The base model then processes q_L from an independent copy of the same shared cache:

$$p_{0,1} = f_\theta\big(q_L \mid \mathrm{copy}(KV_q)\big). \tag{11}$$

Fallback therefore reuses the frozen base prefix rather than repeating query prefill. Attention masks and position indices are preserved across branches, ensuring that each branch continues from the same prompt position.

Training and inference consistency. Cache compatible distillation uses the same boundary during training. The student first computes KV_q with the adapter disabled and then activates the document adapter for q_L and the answer tokens. The distillation objective is evaluated over the supervised answer token distributions. Each adapter is therefore trained under the same frozen base prefix states that it receives during internal routing, Self-Gating, and answer generation.

Teacher cache boundary. The query side cache described above is distinct from the teacher side cache used during memory construction. For a fixed document, the teacher cache contains the document containing prefix shared by its synthetic queries. The query dependent tokens and answer tokens are processed after this boundary. Teacher side cache reuse avoids repeatedly encoding the same document, while student side cache reuse avoids repeatedly encoding the same query for different candidate adapters.

B Selection, Activation, and Cache Ablations

B.1 Router implementation and feature ablations

Feature construction. For a query q, let a_i be the cosine similarity of candidate $i \in \mathcal{S}_k(q)$ and let $a_{\max} = \max_{j \in \mathcal{S}_k(q)} a_j$. The adapter router represents each candidate with

$$x_i = [a_i,\ a_{\max} - a_i,\ H_i,\ h_i]\,, \tag{12}$$

where $a_{\max} - a_i$ encodes the candidate's retrieval rank relative to the strongest retrieved item, H_i is the entropy of its first-answer-token distribution, and h_i is the corresponding hidden representation. The resulting input is standardized using training-set statistics and passed through a two-layer MLP.

Optimization and splits. Training uses weighted binary cross-entropy and AdamW with learning rate 10^{-3}, weight decay 10^{-3}, and 100 optimization steps. We use 100 synthetic questions per NarrativeQA document for router training; the official evaluation questions remain held out.

Feature ablation. Table 6 isolates the contribution of retrieval features and first-token features on NarrativeQA under fixed candidate sets. The results show that both feature groups contribute to router performance. Removing cosine similarity and the similarity gap substantially reduces selection accuracy and downstream ROUGE-L, especially at $k = 5$, where distinguishing among more candidates becomes more difficult without retrieval signals. Removing first-token entropy and hidden-state features causes a smaller but consistent degradation, indicating that model-derived information provides complementary evidence beyond retrieval relevance.

B.2 Self-Gating

Ablation on Self-Gating. A retrieved memory should not always be activated, since an irrelevant adapter may interfere with queries that can be answered by the base model. We evaluate this tradeoff using document-specific NarrativeQA queries and context-independent CommonsenseQA queries.

Table 6: Router feature ablations on NarrativeQA.

Feature set	Sel. acc. @3	R-L @3	Sel. acc. @5	R-L @5
Full features	68.96	23.95	69.27	23.94
w/o retrieval features $(a_i,\ a_{\max} - a_i)$	57.74	21.52	53.13	20.36
w/o first-token features $(H_i,\ h_i)$	65.07	22.83	65.07	22.83

Without Self-Gating, the selected adapter remains active during answer generation. With Self-Gating, the system retains the adapter when its first-token entropy is below the calibrated threshold and otherwise falls back to the base model.

Table 7 reports the results for different candidate-set sizes. Across all three backbones and all values of k, Self-Gating improves CommonsenseQA accuracy by 2.30–8.32 percentage points. On NarrativeQA, the corresponding change ranges from a decrease of 1.26 ROUGE-1 points to an increase of 0.03 points. In particular, Self-Gating slightly improves both NarrativeQA and CommonsenseQA for Qwen2.5-7B, while the other two backbones trade a small decrease in NarrativeQA performance for a larger recovery on CommonsenseQA. Increasing the candidate set from one to five improves the gated NarrativeQA results for all three backbones without substantially reducing CommonsenseQA accuracy. These results show that routing and activation address different decisions: routing selects the most relevant memory among the retrieved candidates, whereas Self-Gating determines whether the selected memory should be used.

Table 7: Gating ablation on NarrativeQA and CommonsenseQA.

Top-k	Gating	NarrativeQA						CommonsenseQA		
		Qwen2.5-0.5B		Qwen2.5-7B		Llama-3.2-1B		Qwen2.5-0.5B	Qwen2.5-7B	Llama-3.2-1B
		R-1 ↑	R-L ↑	R-1 ↑	R-L ↑	R-1 ↑	R-L ↑	Acc. ↑	Acc. ↑	Acc. ↑
1	w/o	23.47	22.54	30.12	29.26	22.85	22.10	37.30%	82.56%	35.95%
	w/	22.74	21.84	30.15	29.28	21.85	21.11	45.62%	84.93%	38.74%
3	w/o	24.47	23.95	32.21	31.32	23.39	22.62	40.54%	82.80%	35.38%
	w/	23.40	22.43	32.22	31.33	22.18	21.42	47.91%	85.18%	38.25%
5	w/o	24.54	23.94	32.61	31.72	23.44	22.66	39.80%	82.64%	36.36%
	w/	23.44	22.48	32.64	31.74	22.18	21.41	47.34%	85.26%	38.66%

Threshold sensitivity. We calibrate Self-Gating using training queries of NarrativeQA and CommonsenseQA. The evaluation questions are therefore not used for threshold selection.

For each query, the retriever first selects the top-1 adapter. Let H_i denote the first-token predictive entropy obtained from the retrieved adapter using the shared-prefix cache. The gate activates the adapter when

$$g_i(\lambda) = \mathbf{1}\{H_i < \lambda\}\,, \tag{13}$$

where $g_i(\lambda) = 1$ means that generation continues with the retrieved adapter. Otherwise, the system falls back to the base model.

NarrativeQA queries are labeled for activation because they require document-specific information, whereas CommonsenseQA queries are labeled for fallback because they are context independent. We define

$$\mathrm{Acc}_{\mathrm{NQA}}(\lambda) = \frac{1}{|\mathcal{D}_{\mathrm{NQA}}|} \sum_{i \in \mathcal{D}_{\mathrm{NQA}}} \mathbf{1}\{H_i < \lambda\}\,, \tag{14}$$

$$\mathrm{Acc}_{\mathrm{CSQA}}(\lambda) = \frac{1}{|\mathcal{D}_{\mathrm{CSQA}}|} \sum_{i \in \mathcal{D}_{\mathrm{CSQA}}} \mathbf{1}\{H_i \geq \lambda\}\,. \tag{15}$$

Their balanced average gives equal weight to activation and fallback accuracy.

We sweep λ from 3.5 to 4.5 in increments of 0.05. Figure 8 shows the expected tradeoff: a higher threshold activates more adapters, increasing NarrativeQA activation accuracy while reducing

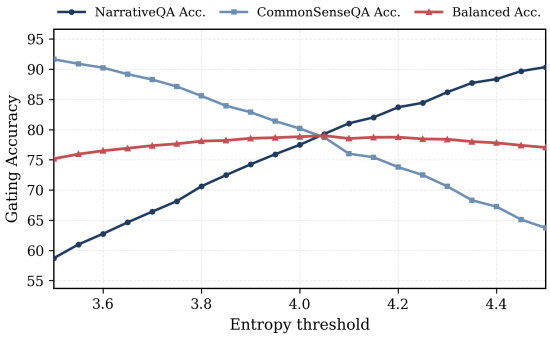


Figure 8: Sensitivity of Self-Gating to the entropy threshold. The threshold controls the tradeoff between activating adapters for NarrativeQA and falling back to the base model for CommonsenseQA, while balanced accuracy remains stable over a broad interval.

Table 8: Base-only entropy classification. The best balanced accuracy is shown in **bold**.

Threshold λ	NQA Acc.	CSQA Acc.	Balanced Acc.
3.00	13.38%	91.65%	52.51%
3.25	20.53%	88.45%	54.49%
3.50	28.07%	84.68%	56.38%
3.75	34.74%	78.95%	**56.84%**
4.00	40.56%	72.81%	56.68%
4.25	46.29%	64.21%	55.25%
4.50	52.77%	55.12%	53.94%
4.75	58.76%	45.70%	52.23%
5.00	65.03%	37.18%	51.11%

CommonsenseQA fallback accuracy. Nevertheless, balanced accuracy remains stable over a broad interval. Between $\lambda = 3.75$ and 4.25, it ranges from 77.65% to 79.00%, with a maximum of 79.00% at $\lambda = 4.05$. Self-Gating is therefore not sensitive to fine-grained threshold tuning within this range.

Controlling for dataset identity. Because the activation and fallback labels are associated with NarrativeQA and CommonsenseQA, respectively, the gate could exploit differences between their query distributions rather than the interaction between a query and its retrieved memory. We measure the strength of this dataset-specific signal using only the Qwen2.5-0.5B base model, without LoRA adapters, retrieved context, or retrieval features. Given the base model's first-token entropy H_i^{base}, we classify a query as NarrativeQA when $H_i^{\text{base}} < \lambda$ and as CommonsenseQA otherwise.

As shown in Table 8, the best balanced accuracy obtained from base-model entropy is only 56.84%. Selecting the threshold on a calibration split and evaluating it on a disjoint holdout split produces a similar balanced accuracy of 57.33%. In comparison, adapter-conditioned Self-Gating reaches 79.00%, an improvement of 22.16 percentage points. These results are consistent with the substantial overlap between the Base distributions in Figure 4. Base-model entropy contains a weak dataset-specific signal, but this signal alone cannot explain the performance of Self-Gating.

B.3 Cache-compatible distillation ablations

We investigate three questions: whether cache-compatible distillation compromises context-specific performance, whether consistency between training and inference is necessary, and whether cache-compatible distillation better preserves general capabilities.

Compatibility with distillation objectives. Our shared-prefix interface is independent of the distillation divergence. We evaluate four representative objectives: forward KL, reverse KL, top-k logit distillation, and an exponential-moving-average (EMA) teacher. Forward KL, used by several context-distillation methods (Cao et al., 2025; Eyuboglu et al., 2025; Li et al., 2026b), minimizes

$$D_{\text{KL}}(\pi_\theta(\cdot \mid q, c) \,\|\, \pi_{\theta+\Delta\theta}(\cdot \mid q)) \,. \tag{16}$$

Table 9: Comparison of standard and cache-compatible prefixes across different distillation objectives. The better average result for each metric is shown in **bold**.

Objective	Prefix	NQA R-1	NQA R-L	SQuAD EM	SQuAD F1
Forward KL	Standard	32.69	31.74	33.16	50.35
	Cache-Compatible	31.23	30.22	33.38	50.05
Reverse KL	Standard	31.97	31.01	33.91	51.60
	Cache-Compatible	30.74	29.73	34.10	51.16
Top-k logits	Standard	30.78	29.81	31.30	49.90
	Cache-Compatible	30.18	29.19	32.25	50.32
EMA teacher	Standard	16.18	15.25	13.17	29.67
	Cache-Compatible	18.57	17.63	16.62	32.09
Average	Standard	**27.91**	**26.95**	27.89	45.38
	Cache-Compatible	27.68	26.69	**29.09**	**45.91**

Reverse KL instead places the student distribution in the first argument, as in on-policy self-distillation (Shenfeld et al., 2026; Hübotter et al., 2026; Zhang et al., 2026):

$$D_{\mathrm{KL}}(\pi_{\theta+\Delta\theta}(\cdot \mid q) \,\|\, \pi_{\theta}(\cdot \mid q, c)) \,. \tag{17}$$

Top-k distillation restricts the divergence to the teacher's highest-probability tokens (Raman et al., 2023; Ye et al., 2026); we use the reverse-KL orientation over the renormalized top-k support. The EMA variant updates the teacher parameters according to

$$\bar{\theta}_t = \alpha \bar{\theta}_{t-1} + (1-\alpha)\theta_t, \tag{18}$$

and computes the forward-KL target from $\pi_{\bar{\theta}_t}(\cdot \mid q, c)$. Table 9 compares two matched pipelines: conventional distillation with adapter-dependent prefixes and cache-compatible distillation with frozen-base shared prefixes during both training and inference. Cache-compatible distillation remains comparable to conventional distillation across all four objectives rather than benefiting only forward KL. Averaged across objectives, NarrativeQA changes by less than 0.3 points, while SQuAD EM and F1 improve by 1.20 and 0.53 points, respectively. These results support compatibility across objectives rather than a uniform accuracy improvement.

Training–inference consistency. We next isolate whether the shared-prefix interface must be used consistently during training and inference. In Table 10, both variants reuse a frozen-base prefix at inference; the only difference is whether the adapter was trained with the same shared-prefix interface. Applying shared-prefix inference to a conventionally trained adapter creates a mismatch between the prefix states observed during training and deployment. Cache-compatible training improves NarrativeQA ROUGE-1 and ROUGE-L by 4.24 and 4.22 points and SQuAD EM and F1 by 6.36 and 6.58 points, respectively. This comparison demonstrates the importance of training–inference consistency and should not be interpreted as evidence that cache reuse alone improves accuracy.

Table 10: Cache-compatible training ablation.

Training interface	NQA R-1	NQA R-L	SQuAD EM	SQuAD F1
Cache-compatible	**31.23**	**30.22**	**33.38**	**50.05**
Conventional	26.99	26.00	27.02	43.47

Specialization and general-capability retention. Table 11 compares the specialization and general-capability retention of different distillation strategies. Both naive and cache-compatible distillation substantially improve context-specific performance over the undistilled model, while cache-compatible distillation remains within -1.46 to $+1.18$ points of naive distillation. Meanwhile, it recovers 11.79–22.03 percentage points of context-agnostic CommonsenseQA accuracy over naive distillation and even surpasses the undistilled model for both Instruct backbones. Thus, cache-compatible distillation retains document-specific specialization while substantially mitigating the general-capability degradation caused by naive distillation.

Table 11: Effect of distillation strategy on specialization and generalization. Context-specific performance is measured using document-level oracle routing, with ROUGE-1 on NarrativeQA and macro F1 on LongBench. Context-agnostic performance is measured by accuracy on CommonsenseQA. Higher is better.

Backbone	Evaluation	Distillation Strategy		
		w/o Distillation	Naive	Cache-Compatible
NarrativeQA				
Qwen2.5-0.5B Base	Context-Specific	13.04	**32.69**	31.23
	Context-Agnostic	**48.48**	24.08	37.30
Llama-3.2-1B Base	Context-Specific	9.20	**33.18**	32.27
	Context-Agnostic	**39.31**	24.16	35.95
LongBench				
Qwen2.5-0.5B-Instruct	Context-Specific	17.04	**23.48**	22.59
	Context-Agnostic	43.82	25.39	**47.42**
Llama-3.2-1B-Instruct	Context-Specific	17.20	27.10	**28.28**
	Context-Agnostic	43.49	28.91	**45.45**

C Discussion of the Cumulative Paradigm

C.1 Idealized recursive interpretation

Let $\mathcal{C} = (c_1, c_2, \ldots, c_n)$ be an ordered stream of contexts. Cumulative methods such as TempLoRA (Wang et al., 2024) and InfiniteICL (Cao et al., 2025) repeatedly update a shared parameter state. At step i, a generic cumulative-distillation objective can be written as

$$\theta_i = \arg\min_{\vartheta} \mathbb{E}_{q\sim\mathcal{Q}_i} D_{\mathrm{KL}}\big(\pi_{\theta_{i-1}}(\cdot \mid c_i, q) \,\big\|\, \pi_{\vartheta}(\cdot \mid q)\big)\,, \tag{19}$$

where θ_{i-1} is the state after processing $c_1, \ldots, c_{i-1}$, and $\mathcal{Q}_i$ is the query distribution used at step i.

To characterize an idealized limit, suppose that optimization and model capacity are sufficient to produce parameters θ_i^* satisfying

$$\pi_{\theta_i^*}(\cdot \mid q) = \pi_{\theta_{i-1}^*}(\cdot \mid c_i, q) \quad \text{for every relevant, possibly augmented prompt } q, \tag{20}$$

with $\theta_0^* = \theta_0$, the base-model parameters. Applying Eq. 20 recursively gives

$$\pi_{\theta_i^*}(\cdot \mid q) = \pi_{\theta_0}(\cdot \mid c_1, c_2, \ldots, c_i, q). \tag{21}$$

Thus, under this strong assumption, the cumulative state reproduces the base model conditioned on the concatenation of all contexts observed so far. Equation 21 is an equivalence of predictive distributions, not an upper bound on downstream task accuracy: finite capacity, optimization error, and limited synthetic-query coverage can all create an additional gap.

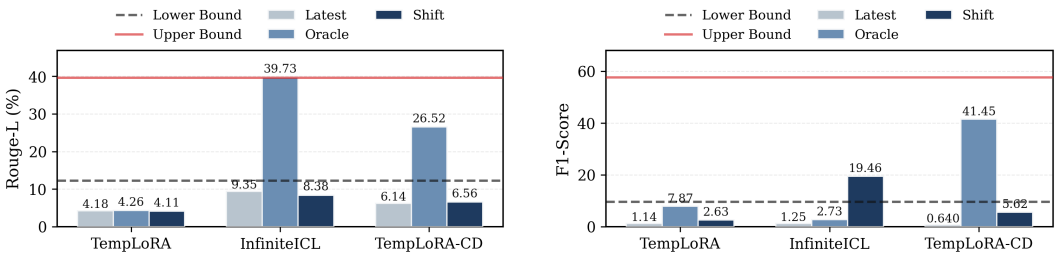
This interpretation exposes two limitations. First, the distilled state inherits the behavior of a teacher conditioned on an increasingly long context, including any degradation caused by long-range distraction or positional effects. Second, every context is written into the same state without an explicit selection mechanism. Irrelevant or conflicting contexts can therefore interfere with information needed by later queries. Independent memories trade a larger storage footprint for the ability to retrieve and activate individual document memories.

C.2 Evaluation settings

Let $\Delta\theta_i$ denote the cumulative adapter state after processing $c_1, \ldots, c_i$, rather than a document-specific adapter for c_i. For a query q_i associated with context c_i, we use the following settings.

LoRA-only cumulative memory. All contextual information must be recovered from the cumulative adapter state:

- ***Latest*** uses the final state, $f_{\theta+\Delta\theta_n}(q_i)$.
- ***Oracle*** uses the state immediately after c_i is written, $f_{\theta+\Delta\theta_i}(q_i)$. This is a step oracle used for diagnosis, not a deployable selection rule.



(a) NarrativeQA. (b) SQuAD.

Figure 9: Cumulative-memory results with Qwen2.5-0.5B under Latest, Oracle, and Shift evaluation.

- ***Shift*** uses the state after one additional update, $f_{\theta+\Delta\theta_{i+1}}(q_i)$, for $i < n$. It measures whether writing c_{i+1} disrupts information about c_i.

LoRA plus explicit context. InfiniteICL combines a cumulative adapter as long-term memory with one explicit context as short-term memory:

- ***Latest*** uses the most recent pair, $f_{\theta+\Delta\theta_{n-1}}(q_i, c_n)$.
- ***Oracle*** uses the pair available when c_i is written, $f_{\theta+\Delta\theta_{i-1}}(q_i, c_i)$.
- ***Shift*** advances both components by one step, $f_{\theta+\Delta\theta_i}(q_i, c_{i+1})$, for $i < n$.

These settings separate retention across updates from access to the explicit context. Because the LoRA-plus-context settings change both components together, they diagnose sensitivity to temporal alignment but do not by themselves identify which component produced an answer.

C.3 Empirical analysis

We evaluate the cumulative baselines with Qwen2.5-0.5B on NarrativeQA and SQuAD. Figure 9 compares the three settings.

Sensitivity to subsequent updates. Performance drops substantially when a query is evaluated after the cumulative state has advanced. Under *Shift*, TempLoRA-CD reaches 6.56 ROUGE-L on NarrativeQA and 5.62 F1 on SQuAD, despite being substantially stronger under *Oracle*. Because *Shift* differs from *Oracle* by a single additional update, this gap directly measures interference from writing the next context; it is not caused by a retrieval algorithm.

Gap to independent memories. Even with step-oracle access, a cumulative state must preserve multiple contexts in shared parameters. TempLoRA-CD obtains 26.52 ROUGE-L on NarrativeQA and 41.45 F1 on SQuAD, compared with 30.22 and 50.05 for our independent-memory method in Table 10. This result is consistent with interference from accumulated contexts, although it does not by itself separate capacity, optimization, and data-order effects.

Entanglement of latent and explicit memory. InfiniteICL changes both its cumulative adapter and explicit context across the settings. On NarrativeQA, ROUGE-L decreases from 39.73 under *Oracle* to 8.38 under *Shift*; on SQuAD, F1 increases from 2.73 to 19.46. The opposite trends show that the relative contribution of the adapter and explicit context is dataset dependent and cannot be inferred from the combined prediction alone. These results motivate explicit memory selection and activation, for which the contribution of each candidate memory can be evaluated directly.

D Efficiency Measurement Details

D.1 Training

We measure training efficiency using Qwen2.5-0.5B (Yang et al., 2024) in bfloat16 precision with a batch size of 16 and 1,000 NarrativeQA examples (Kočiskỳ et al., 2018). We evaluate context lengths

from 500 to 16,000 tokens, with each target answer fixed to 32 tokens. The reported construction cost includes teacher-answer generation, one-time KV-cache precomputation where applicable, and the complete adapter-training loop. Runtime is measured as wall-clock time, peak memory is recorded using CUDA peak-memory statistics, and FLOPs are estimated using DeepSpeed FlopsProfiler. Synthetic-question generation is excluded because the same questions are shared across all compared methods. The training results are reported in Figure 5.

We further evaluate how many queries are required to amortize the one-time memory-construction cost. For each context length, the break-even query count is computed as

$$N_{\mathrm{BE}} = \left\lceil \frac{C_{\mathrm{build}}}{C_{\mathrm{text}} - C_{\mathrm{memory}}} \right\rceil, \qquad C_{\mathrm{text}} > C_{\mathrm{memory}}, \tag{22}$$

where C_{build} is the measured memory-construction cost, and C_{text} and C_{memory} are the per-query costs of textual RAG and latent-memory inference, respectively. The per-query costs use the same 16-token answer-generation setting as the inference benchmark. As shown in Figure 7, textual RAG is cheaper below each curve, whereas latent-memory inference becomes cheaper once the number of queries exceeds the corresponding break-even point. The break-even count decreases for longer contexts because textual RAG repeatedly encodes the retrieved document, while latent-memory inference amortizes the one-time construction cost across queries.

D.2 Inference

We measure inference efficiency using 100 NarrativeQA queries, with each method generating 16 answer tokens. The reported FLOPs include query-prefix prefill, candidate routing or gating, and final answer generation. For cache-compatible methods, we account for both the initial frozen-base prefix prefill and subsequent adapter-specific cache-reuse operations. The results are reported in Figure 6.

E Scalability of Self-Gating and Routing

Table 12: Scalability of Self-Gating across memory-bank sizes.

#Adapters	NQA Acc.	CSQA Acc.	Balanced Acc.
50	74.03%	82.72%	78.38%
100	72.91%	84.36%	**78.64%**
200	73.81%	83.37%	78.59%
355	74.51%	81.90%	78.20%

Scalability of Self-Gating. Table 12 shows that Self-Gating remains stable as the memory bank grows from 50 to 355 adapters. Across all bank sizes, the NQA activation accuracy stays between 72.91% and 74.51%, while the CSQA fallback accuracy remains between 81.90% and 84.36%. Consequently, the balanced accuracy varies by only 0.44 percentage points, from 78.20% to 78.64%, with the best result obtained at 100 adapters. This limited variation indicates that increasing the number of stored latent memories does not substantially degrade the gate's ability to distinguish context-specific queries from context-independent ones. Overall, Self-Gating scales robustly with the memory-bank size and maintains a consistent balance between activating useful memories and rejecting irrelevant ones.

Table 13: Scalability of internal routing across memory-bank sizes.

#Adapters	Top-1			Top-3			Top-5		
	Routing Acc.	R-1	R-L	Routing Acc.	R-1	R-L	Routing Acc.	R-1	R-L
50	69.49%	24.98	24.03	76.90%	26.69	25.72	**78.64%**	**27.13**	**26.14**
100	67.09%	24.24	23.38	73.01%	25.79	24.92	**74.23%**	**25.96**	**25.05**
200	66.21%	23.91	23.02	70.55%	**25.09**	**24.16**	**70.85%**	25.02	24.07
355	65.07%	23.47	22.54	68.96%	24.47	**23.95**	**69.27%**	**24.54**	23.94

Scalability of internal routing. Table 13 shows that internal routing consistently improves over Top-1 retrieval across different memory-bank sizes. Across memory banks ranging from 50 to 355 adapters, both Top-3 and Top-5 routing consistently outperform Top-1 retrieval in selection accuracy and downstream NarrativeQA performance. At 355 adapters, Top-5 retains a 4.20-point selection-accuracy advantage over Top-1.

Compared with Top-1, Top-5 improves routing accuracy by 9.15, 7.14, 4.64, and 4.20 percentage points for bank sizes of 50, 100, 200, and 355, respectively. These improvements also translate into consistent gains in ROUGE-1 and ROUGE-L. Top-3 exhibits the same overall trend and slightly outperforms Top-5 in ROUGE scores at 200 adapters. Overall, the results demonstrate that internal routing remains effective across different memory-bank scales and continues to provide reliable improvements as the number of stored latent memories increases.