Context Distillation as Latent Memory Management

Anonymous ACL submission

Abstract

Context distillation compresses contextual information into model parameters, yet existing methods often ignore how multiple distilled latent memories should be stored, retrieved, and safely activated in non-oracle settings. We formulate context distillation as a latent memory management problem. We distill each context into an independent LoRA adapter, forming a modular memory bank that enables explicit memory selection. Given a query, our framework retrieves candidate memories, routes the query to the most suitable adapter, and uses a Self-Gating mechanism to decide whether latent memory should be activated. To improve efficiency, we further introduce cache sharing to reduce management overhead during inference. Experiments show that our method substantially outperforms baselines with retrieval, while Self-Gating improves robustness by deactivate unnecessary latent memories.

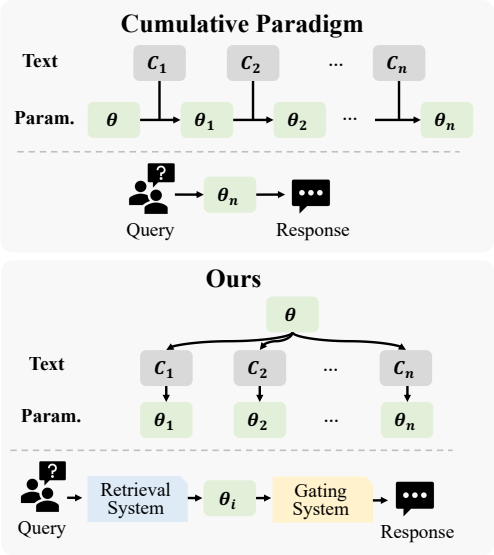


Figure 1: Comparison of cumulative context distillation and our method. Cumulative distillation compresses all contexts into one adapter, whereas our framework stores contexts as *modular memories*: *retrieves, routes*, and *gates* the latent memories for each query.

1 Introduction

LLMs adapt to documents, tasks, and users primarily by placing relevant information in the context window (Brown et al., 2020; Lewis et al., 2020). This in-context adaptation is flexible but inherently temporary: the same information must be re-read for every query, and long prompts increase latency (Tay et al., 2022), memory usage, and generation instability (Zhao et al., 2021; Liu et al., 2024). These limitations raise a central question: *Can contextual information be converted from temporary text into persistent model memory, and if so, how should many such memories be stored, retrieved, and safely activated?*

Context distillation (CD) (Cao et al., 2025; Wang et al., 2024; Ye et al., 2026; Zhang et al., 2026; Charakorn et al., 2026, 2025) offers a natural answer to the first part of this question. Instead of conditioning on a long context at every inference step, CD trains a model to reproduce its context-conditioned behavior without explicitly seeing the context, thereby compressing textual context into latent memory stored in model parameters for future use. However, the second part of the question remains largely unaddressed. Practical deployment requires not only writing a context into parameters, but also managing a collection of parameterized memories. Existing CD methods typically bypass this issue by assuming oracle access to the relevant distilled memory (Ye et al., 2026; Zhang et al., 2026; Snell et al., 2022; Caccia et al., 2025), or by continually writing new contexts into a single adapter with a cumulative paradigm (Wang et al., 2024; Cao et al., 2025).

The cumulative paradigm, as illustrated in Figure 1, is conceptually simple: each new context is absorbed into the current parameter state, so the resulting model is expected to implicitly contain all previously observed contexts. As a result, newly distilled contexts can overwrite earlier ones (Shenfeld et al., 2026; Li and Hoiem, 2017; Kirkpatrick

et al., 2017), leading to severe forgetting and unreliable activation of previously latent memories. Moreover, since all contexts are compressed into a single state, the model lacks an explicit mechanism to select among memories or to deactivate them when a query does not require contextual knowledge. Consequently, cumulative distillation is fragile in realistic non-oracle settings, where the system must determine both which latent memory to use and whether any memory should be activated.

In this paper, we formulate context distillation as a *latent memory management* problem, where distilled contexts are maintained as persistent modular memories. Specifically, we distill each context into a dedicated LoRA adapter to form a memory bank. As shown in Figure 1, our framework retrieves candidate memories for a given query, routes it to the most suitable adapter, and employs a Self-Gating mechanism to decide whether latent memory should be activated, avoiding harmful use on context-agnostic queries. To improve efficiency, we further introduce cache-sharing to amortize context encoding during distillation and reduce retrieval and gating overhead at inference time. Experiments under realistic settings show that our method substantially outperforms cumulative-memory baselines, while Self-Gating and cache-sharing improve robustness and efficiency.

Our contributions are threefold:

- We formulate context distillation as latent memory management, shifting focus from single-context internalization to storage, retrieval, and activation of latent memories.
- We propose a modular adapter memory bank with two-stage latent retrieval and Self-Gating.
- We introduce cache-sharing distillation, enabling efficient adapter switching while preserving downstream performance.

2 Background

2.1 KV-cache Compression

KV-cache compression aims to compress long contexts into shorter latent tokens, typically achieving a 15×-30× compression ratio. Previous works (Eyuboglu et al., 2025; Liu and Qiu, 2025; Li et al., 2026) first compress the context c with a compressor ϕ and apply context distillation target:

$$D_{\mathrm{KL}}(\pi(\cdot \mid q, c) \,\|\, \pi(\cdot \mid q, \phi(c)))\,, \qquad (1)$$

where q denotes a query. The selection of ϕ varies across different works. Cartridges (Eyuboglu et al., 2025) and Latent Context Compilation (Li et al., 2026) compress context via Base Model with LoRA, while C3 (Liu and Qiu, 2025) uses a small Base Model as context compression encoder.

2.2 Context Distillation

Context distillation aims to internalize the context into the model parameters by minimizing:

$$D_{\mathrm{KL}}(\pi_\theta(\cdot \mid q, c) \,\|\, \pi_{\theta+\Delta\theta}(\cdot \mid q))\,. \qquad (2)$$

Snell et al. (2022) first proposed internalizing contextual information into model parameters through distillation. Similarly, Prompt baking (Bhargava et al., 2024) proposes baking prompt information into model parameters. Charakorn et al. (2025, 2026) adopt the idea of meta-learning and use a hypernetwork to facilitate the distillation. Recently, OPCD (Ye et al., 2026) and Opsdl (Zhang et al., 2026) adopt on-policy distillation to further enhance the distillation capability. Caccia et al. (2025) first discuss the synergies between CD and RAG. However, they only augment the oracle latent memory with the retrieved passage, instead of managing the latent memory itself. Beyond internalizing a single context, how to manage latent memories remains largely unaddressed.

2.3 Cumulative Distillation

Cumulative context distillation (Wang et al., 2024; Cao et al., 2025) updates the model sequentially by distilling each new context into its parameters:

$$\theta_i = \arg\min_{\theta} D_{\mathrm{KL}}\big(\pi_{\theta_{i-1}}(\cdot \mid q, c_i) \,\|\, \pi_\theta(\cdot \mid q)\big), \qquad (3)$$

with the goal of progressively internalizing all preceding contexts into the latest model θ_n, thereby enabling inference with the latest adapter instead of relying on an oracle selection of the appropriate adapter. However, such sequential writing can lead to severe catastrophic forgetting (Shenfeld et al., 2026; Li and Hoiem, 2017; Kirkpatrick et al., 2017). Beyond the forgetting issue, we further argue that under an idealized recursive distillation assumption, cumulative distillation behaves analogously to distilling the Base Model conditioned on accumulated contexts $\{c_1, \ldots, c_i\}$:

$$\pi_{\theta_i^*}(y \mid q) = \pi_\theta(y \mid q, [c_1, \ldots, c_i]). \qquad (4)$$

We provide further details in Appendix A. This perspective suggests that cumulative CD inherits the teacher model's sensitivity to irrelevant accumulated contexts, as well as its degradation under long-context inputs. Consequently, performance may deteriorate even under the oracle test setting.

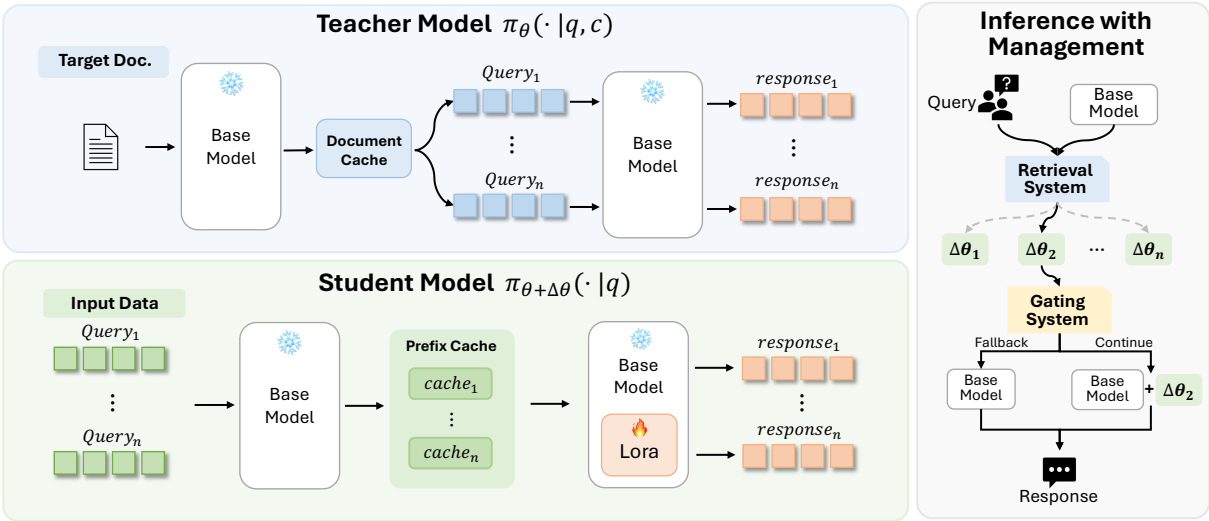


Figure 2: Overview of our training stage and inference stage. **Training:** Teacher model precomputes the KV-cache of target document for reuse during training, while student computes the prefix KV-cache with Base Model to enable the *Hot-swap* property for management. **Inference:** Given a query, we first retrieve the most relevant latent memory $\Delta\theta_i$, and then use Self-Gating to decide whether to continue with $\theta + \Delta\theta_i$ or fallback to θ. Owing to the cache-sharing distillation introduced during training, memory management remains efficient.

3 Methodology

3.1 Cache-Sharing Context Distillation

Given a context stream $\mathcal{C} = \{c_1, c_2, \ldots, c_t\}$, we freeze the base model f_θ and train a separate PEFT module, e.g., LoRA, $\Delta\theta_i$ for each context c_i. The student model $\theta + \Delta\theta_i$ is trained to mimic the teacher model, i.e., the base model explicitly conditioned on c_i. For a set of queries $\{q_1, \ldots, q_n\}$, we minimize the KL divergence:

$$\Delta\theta_i^* = \arg\min_{\Delta\theta_i} \frac{1}{n} \sum_{j=1}^{n} D_{\mathrm{KL}}\big(\pi_\theta(y_j|q_j, c_i) \parallel \pi_{\theta+\Delta\theta_i}(y_j|\mathrm{KV}_{q_j})\big), \tag{5}$$

where $\mathrm{KV}_{q_j} = f_\theta(q_j)$ and y_j denotes the target response distribution produced by the teacher. To reduce both training and inference cost, we introduce a dual cache-sharing mechanism for the teacher and student models.

Teacher Cache For the teacher $\pi_\theta(\cdot|q_j, c_i)$, prefilling the long context c_i dominates computation. During distillation of a fixed adapter $\Delta\theta_i$, however, c_i remains unchanged, while only q_j and y_j vary. We therefore precompute KV-cache of c_i once with the base model and reuse it for all training steps, substantially accelerating teacher-side generation.

Student Cache We further introduce a decoupled cache-sharing pipeline for the student, yielding nearly zero-overhead adapter switching, which we refer to as the ***Hot-swap*** property. Specifically, student-side cache sharing deliberately trains a cache-compatible student distribution. Instead of applying the adapter while encoding the query prefix, we compute the prefix KV-cache with the frozen base model and activate the adapter only for response generation. This restricts the adapter to be compatible with base-model prefix caches, enabling adapter hot-swapping without KV recomputation. Consequently, the system can seamlessly switch among adapters $\Delta\theta_i$ while reusing the same prefix cache, which is essential for the Gating and Retrieval systems described in Sections 3.2 and 3.3.

3.2 Self-Gating System

As illustrated in Figure 4, we observe a distinct phenomenon in the predictive entropy of the first generated token. In standard In-Context Learning (ICL), there exists a clear distributional margin between context-specific queries and context-agnostic queries. In contrast, the base model without any appended context exhibits no such distributional gap. Since our distillation objective in Eq. 5 explicitly aligns the output distribution of the LoRA student with the ICL teacher, the trained context-specific LoRA models inherit this entropy gap.

Computing this first-token entropy with standard ICL is prohibitively expensive due to long-context. However, our cache-sharing mechanism makes it computationally trivial. Motivated by this observation, we propose a *Self-Gating* mechanism that

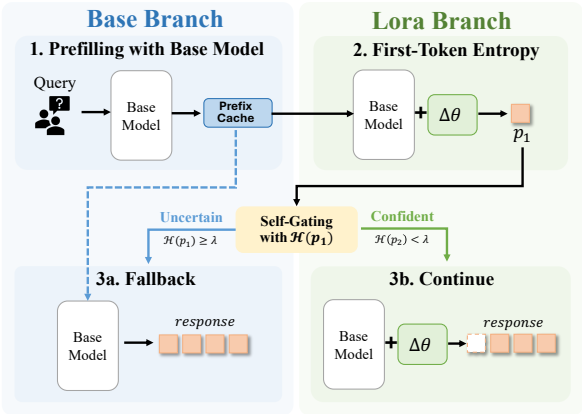


Figure 3: Overview of the proposed Self-Gating mechanism. The system routes the generation between the specific LoRA and the base model based on the first-token entropy, with zero KV-cache recomputation.

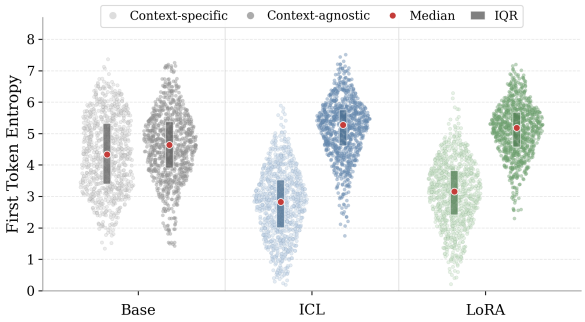


Figure 4: Distribution of the first-token entropy. A clear gap exists between context-specific and context-agnostic queries when context is provided, whereas the base model exhibits no such distinction.

leverages the first-token entropy for dynamic model gating. As shown in Figure 3, given a context adapter $\Delta\theta_i$, we first process the input prefix (i.e., prompt and query, denoted simply as q), exclusively through the frozen base model f_θ to obtain the shared prefix KV-cache:

$$\mathrm{KV}_q = f_\theta(q). \tag{6}$$

We then temporarily activate the LoRA module $\Delta\theta_i$ to generate the probability distribution p_1 of the first token:

$$p_1 = f_{\theta+\Delta\theta_i}(\mathrm{KV}_q). \tag{7}$$

Let $\mathcal{H}(p_1)$ denote the Shannon entropy of this distribution. We use a threshold λ for gating:

- **Continue ($\mathcal{H}(p_1) < \lambda$):** Low entropy indicates that the LoRA model is highly confident in answering the query based on its injected knowledge. We therefore retain the LoRA module and continue decoding.
- **Fallback ($\mathcal{H}(p_1) \geq \lambda$):** High entropy indicates significant uncertainty, suggesting that the query is likely context-agnostic, e.g., a general knowledge question. We then instantly deactivate the LoRA module and fall back to the base model.

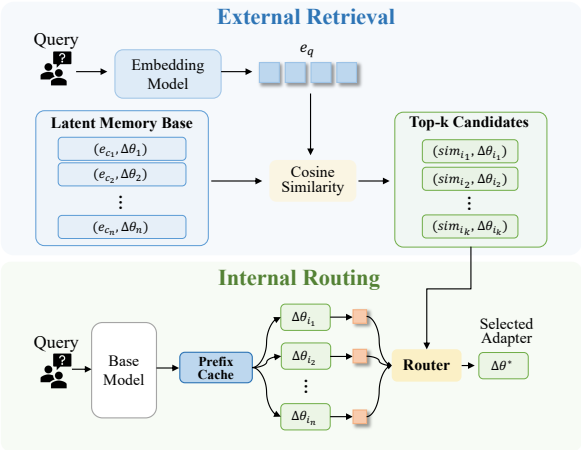


Figure 5: Latent memory retrieval with independently distilled LoRA adapters. An external retriever first selects candidate adapters by cosine similarity, and an internal router further ranks them using first-token entropy and hidden states under a shared prefix KV-cache.

When falling back, the base model seamlessly reuses the already computed KV_q for subsequent generation. Therefore, the *only* additional computational overhead introduced by Self-Gating is the forward pass of a single token using the LoRA model in Eq. 7, making the entire gating process exceptionally efficient.

3.3 Latent Memory Retrieval System

By distilling each context into an independent LoRA module $\{\Delta\theta_1, \ldots, \Delta\theta_t\}$, we recast context utilization as a module selection and routing problem. To manage these distributed latent memories, we introduce a two-stage retrieval system: *External Retrieval* and *Internal Routing*, as shown in Figure 5.

External Retrieval (Coarse-Grained) Similar to dense retrieval in RAG, we use an off-the-shelf embedding model f_{emb} to encode each context and the incoming query:

$$\mathbf{e}_{c_i} = f_{\text{emb}}(c_i), \tag{8}$$

$$\mathbf{e}_q = f_{\text{emb}}(q). \tag{9}$$

We then select the top-K LoRA modules with the highest cosine similarity to the query:

$$\mathcal{S}_{\text{topK}} = \underset{i\in\{1,\ldots,t\}}{\text{TopK}} \left(\cos(\mathbf{e}_{c_i}, \mathbf{e}_q)\right). \tag{10}$$

Internal Routing (Fine-Grained) Although external embeddings capture semantic relevance, they may not align perfectly with the LLM generative confidence. We therefore refine the candidate set $\mathcal{S}_{\text{topK}}$ with an internal router. Using the cache-sharing mechanism in Section 3.2, we first compute the shared prefix KV-cache with the base model. For each candidate adapter $\Delta\theta_i \in \mathcal{S}_{\text{topK}}$, we briefly activate it to obtain the first-token hidden state h_i and predictive entropy $\mathcal{H}_i$. The final adapter is selected by a lightweight feed-forward network(FFN) router conditioned on the external similarity $sim_i = \cos(\mathbf{e}_q, \mathbf{e}_{c_i})$, entropy $\mathcal{H}_i$, and hidden state h_i (details in Appendix C.5):

$$\Delta\theta^* = \arg \max_{\Delta\theta_i \in \mathcal{S}_{\text{topK}}} router(sim_i, \mathcal{H}_i, h_i). \quad (11)$$

Since all candidates share the same query-prefix KV-cache, evaluating multiple LoRAs during internal routing introduces negligible overhead.

4 Experiments

In this section, we empirically study the central question: *How should latent memories be stored, retrieved, and safely activated?*

4.1 Implementation Details

Base Model We conduct experiments with Qwen2.5-0.5B (Yang et al., 2024) and Qwen2.5-7B (Yang et al., 2024) as base models. We use Qwen3-Embedding-0.6B (Zhang et al., 2025) for retrieval.

Benchmarks For context-specific queries, we evaluate on NarrativeQA (Kočiskỳ et al., 2018) and SQuAD (Rajpurkar et al., 2016). For context-agnostic queries, we use CommonsenseQA (Talmor et al., 2019). Since SQuAD contains a large number of documents, we retain the top 300 most frequently queried documents, ranked by the number of associated queries. For the statistics of our dataset, please refer to Appendix B.

Baselines We compare with two cumulative-memory baselines, TempLora (Wang et al., 2024) and InfiniteICL (Cao et al., 2025). Since TempLora does not use QA distillation, we also include TempLoraCD, which replaces its original objective with distillation loss for a fairer comparison.

More details can be found in Appendix C.

4.2 How to Store

In this section, we evaluate cumulative paradigm with different settings. The *Latest* setting always uses the latest adapter $\Delta\theta_n$, the *Shift* setting evaluates a query q^i from document c_i with the next adapter $\Delta\theta_{i+1}$, and *Oracle* settings assume access to the correct context or adapter. More details can be found in Appendix C.4.

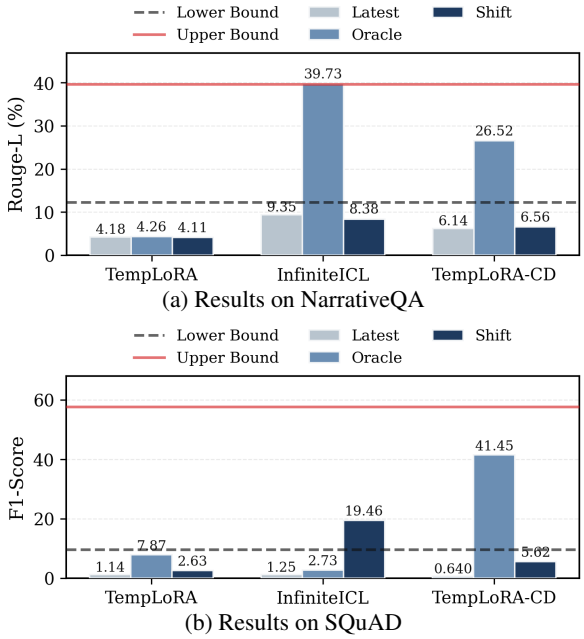


Figure 6: Results of cumulative methods on Qwen2.5-0.5B, under various evaluation settings.

We discuss three key limitations of the cumulative paradigm. First, the cumulative paradigm suffers from severe forgetting in realistic settings. Cumulative methods degrade substantially in both *Latest* and *Shift* settings. For example, TempLoraCD drops to 6.56 ROUGE-L on NarrativeQA and 5.62 EM on SQuAD under *Shift* setting, despite much stronger performance in the *Oracle* setting. This indicates that despite the cumulative distillation, the latent memory is still highly sensitive to adapter selection and prone to retrieval errors.

Second, the cumulative paradigm limits the upper bound of context distillation. Even under *Oracle* setting, cumulative training is affected by increasing cumulative context length and accumulated noise, as discussed in Section 2.3. As a result, TempLoraCD underperforms our method in Table 1: on NarrativeQA, it achieves 26.52 ROUGE-L compared to 30.22 for ours; on SQuAD, it achieves 41.45 F1-Score compared to 50.05 for ours. This suggests that accumulating contexts weakens an adapter's ability to preserve context-specific knowledge, as discussed in Section 2.3.

Last, the cumulative paradigm fails to manage the memory usage. InfiniteICL shows unstable behavior: it is unclear whether predictions come

Table 1: Main results on NarrativeQA and SQuAD. † denotes Oracle settings with access to the ground-truth context or adapter, serving as upper bounds for the corresponding methods.

Method	NarrativeQA				SQuAD			
	QWEN2.5-0.5B		QWEN2.5-7B		QWEN2.5-0.5B		QWEN2.5-7B	
	ROUGE-1 ↑	ROUGE-L ↑	ROUGE-1 ↑	ROUGE-L ↑	EM ↑	F1 ↑	EM ↑	F1 ↑
Lower Bound								
Base Model	13.08	12.26	14.95	14.40	2.38	9.63	13.51	21.97
Upper Bound								
In-context†	40.65	39.64	55.99	55.03	41.98	57.67	78.20	85.40
RAG								
– @3	21.12	20.21	47.36	46.61	26.84	42.61	63.89	72.18
– @5	16.66	15.79	35.57	34.87	26.04	41.15	65.18	73.37
Cumulative Paradigm								
TempLora	4.67	4.18	4.83	4.11	0.00	1.14	0.00	1.60
InfiniteICL	10.10	9.35	4.24	3.77	0.76	1.25	0.91	0.91
TempLoraCD	6.65	6.14	4.09	3.78	0.00	0.64	0.00	0.00
Ours								
Oracle†	31.23	30.22	37.55	36.78	33.38	50.05	43.32	56.97
Retrieval								
– @3	24.54	23.57	28.46	27.68	**28.30**	43.11	**36.34**	**46.57**
– @5	**24.44**	**23.46**	**28.64**	**27.86**	28.14	**43.25**	36.30	46.45

from latent adapter memory or explicit context. For a query q^i from document c_i, InfiniteICL uses $\Delta\theta_{i-1}$ as long-term memory and c_i as short-term memory. On NarrativeQA, it relys more on short-term context, performing well in the *Oracle* setting but dropping from 39.73 to 8.38 ROUGE-L under *Shift*. On SQuAD, the trend is reversed: F1-Score increases from 2.73 to 19.46 under *Shift*, indicating greater reliance on latent memory in adapter. Such results demonstrate that InfiniteICL lacks a stable mechanism to control when to use latent memory versus explicit context. These results motivate explicit adapter management, rather than relying on a single cumulatively updated memory.

4.3 Which to Use

The results in Section 4.2 show that cumulative memories are difficult to use when the correct adapter is unknown. We therefore evaluate whether our proposed retrieval system can make latent memories practical at inference time.

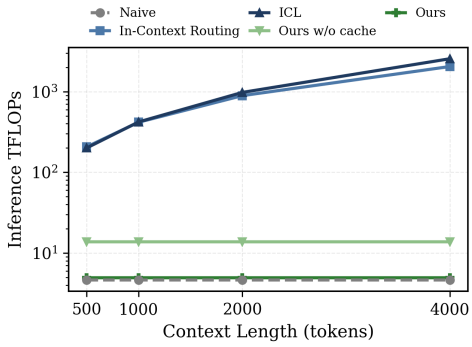
As shown in Table 1, retrieved adapters consistently outperform cumulative baselines under realistic settings. On NarrativeQA, our method achieves 24.54 ROUGE-1 with QWEN2.5-0.5B and 28.64 ROUGE-1 with QWEN2.5-7B, substantially higher than the *latest* variants of TempLora, InfiniteICL, and TempLoraCD. On SQuAD, our Retrieval@3 setting reaches 28.30 EM / 43.11 F1 with QWEN2.5-0.5B, and 36.34 EM / 46.57 F1 with QWEN2.5-7B. These results indicate that independently stored adapters can function as useful latent memories when paired with a retrieval system.

We further compare our method with the oracle setting. Although the retrieved adapters do not always match oracle performance, they provide a deployable alternative when the query–adapter correspondence is unknown. Moreover, compared with the cumulative method, whose performance drops significantly when moving from the oracle to the realistic setting, our retrieval system preserves a substantial fraction of oracle performance while removing the need for access to the oracle adapter.

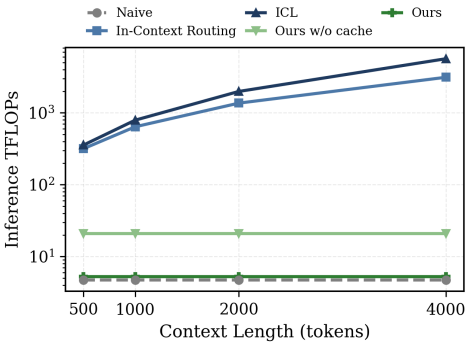
Efficiency Compared with standard textual RAG, our goal is not to universally outperform explicit retrieval in task performance, but to build a retrieval system that manages latent memories and makes them effective in realistic settings at low cost. Figure 7 compares inference efficiency under different retrieval settings. *ICL* denotes generation with concatenated retrieved contexts, while *naive* denotes LoRA-based generation without management. *In-Context Routing* routes using the first token generated with context, and *Ours w/o cache* routes with LoRA but without the shared prefix cache. More details are provided in Appendix E. Explicit in-context methods process retrieved contexts as part

Table 2: Results of hybrid queries from NarrativeQA and CommonsenseQA. Best overall results are shown in **bold**, and second-best results are underlined.

Setting	NarrativeQA				CommonsenseQA	
	QWEN2.5-0.5B		QWEN2.5-7B		QWEN2.5-0.5B	QWEN2.5-7B
	ROUGE-1 ↑	ROUGE-L ↑	ROUGE-1 ↑	ROUGE-L ↑	Accuracy ↑	Accuracy ↑
Base Model	13.08	12.26	14.95	14.40	**47.80%**	**85.18%**
TempLora	4.67	4.18	4.83	4.11	20.56%	20.31%
InfiniteICL	10.10	9.35	3.93	3.56	19.49%	20.56%
TempLoraCD	6.65	6.14	0.75	0.67	19.57%	18.92%
Ours w/o Gating	**23.47**	**22.54**	26.36	25.67	37.30%	72.40%
Ours	22.74	21.84	**26.98**	**26.30**	45.62%	79.44%



(a) Top-3



(b) Top-5

Figure 7: Inference efficiency comparison for retrieval.

of the input, causing FLOPs to grow rapidly with context length. In contrast, our method switches LoRA adapters with the shared prefix cache, avoiding repeated computation over retrieved contexts and keeping the cost close to naive generation. At 4000 tokens, our method reduces FLOPs by $414.7\times$ under Top-3 and $1081.4\times$ under Top-5 compared with explicit in-context inference. Cache reuse further brings a $2.8\times$ and $4.0\times$ reduction over the w/o-cache variant under Top-3 and Top-5, respectively.

4.4 Whether to Activate

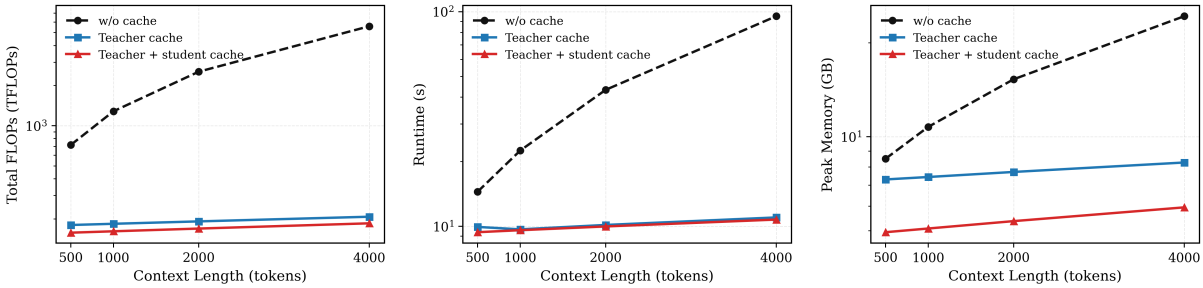
The results in Section 4.3 show that retrieved adapters can serve as effective latent memories for context-specific queries. However, practical systems must also decide *whether* to activate a retrieved memory, since irrelevant adapters may interfere with the base model's general knowledge on context-agnostic queries. We evaluate this issue in a hybrid setting that combines context-specific queries from NarrativeQA and context-agnostic queries from CommonsenseQA, where Self-Gating decides whether to activate the adapter. For isolating activation control, Table 2 evaluates Self-Gating on the top-1 retrieved adapter. This isolates whether the system can suppress an irrelevant latent memory, independent of the internal routing.

As shown in Table 2, always activating an adapter substantially degrades CommonsenseQA performance. Cumulative methods using the *latest* adapter suffer large drops, while our retrieval-only variant reduces CommonsenseQA accuracy from 47.80% to 37.30% on QWEN2.5-0.5B, and from 85.18% to 72.40% on QWEN2.5-7B. These results indicate that retrieval alone is insufficient: retrieved latent memories must still be selectively activated. With gating, our method largely mitigates this interference. CommonsenseQA accuracy improves to 45.62% on QWEN2.5-0.5B and 79.44% on QWEN2.5-7B, while maintaining competitive NarrativeQA performance. These results show that our proposed Self-Gating mechanism enables the model to use latent memory only when necessary and fall back to the base model otherwise.

Efficiency Figure 9 compares the inference cost of different activation strategies. Following the previous setting, *In-Context Gating* denotes gating based on the first token generated with the retrieved context. Unlike in-context methods, whose cost grows with the context length, our method activates memories by selectively enabling LoRA adapters and reusing the shared prefix cache. At 4000 tokens, it reduces FLOPs by $210.0\times$ over in-context gating and $116.6\times$ over ICL, while introducing only about 1% overhead over naive generation.

4.5 Training Efficiency

While CD offers an effective way to reduce inference cost, its training cost increases with the



(a) Flops Comparison (b) Runtime Comparison (c) Memory Comparison

Figure 8: Training efficiency comparison under varying context lengths.

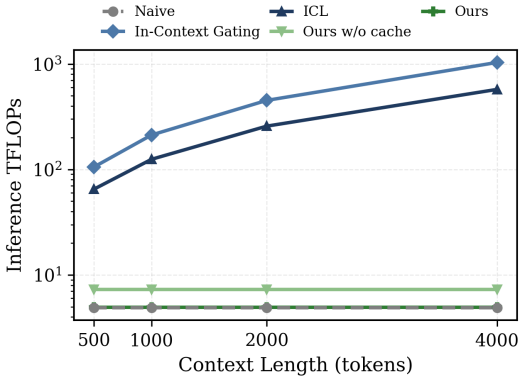


Figure 9: Inference efficiency comparison for gating.

Table 3: Comparison of performance on NarrativeQA and SQuAD with Qwen2.5-0.5B.

Method	NarrativeQA		SQuAD	
	ROUGE-1 ↑	ROUGE-L ↑	EM ↑	F1 ↑
Forward KL	32.69	31.74	33.16	50.35
– W/ CACHE	31.23	30.22	33.38	50.05
Reverse KL	31.97	31.01	33.91	51.60
– W/ CACHE	30.74	29.73	34.10	51.16
Top-k Logits	30.78	29.81	31.30	49.90
– W/ CACHE	30.18	29.19	32.25	50.32
EMA Teacher	16.18	15.25	13.17	29.67
– W/ CACHE	18.57	17.63	16.62	32.09
Average	**27.91**	**26.95**	27.89	45.38
– W/ CACHE	27.68	26.69	**29.09**	**45.91**

context length (Charakorn et al., 2026; Wang et al., 2024; Cao et al., 2025). We evaluate training efficiency by varying the context length from 500 to 4000 tokens. As shown in Figure 8, compared with the w/o cache variant that repeatedly recomputes the full context, teacher-side cache reuse and student-side prefix caching substantially reduce computation. At the longest context length, the full caching pipeline achieves an $8.4\times$ speedup, a $4.1\times$ reduction in peak memory usage, and a $21.9\times$ reduction in FLOPs. Moreover, whereas the baseline cost grows rapidly with context length, our cached pipeline keeps training time nearly constant, indicating that the cost of long-context processing is effectively amortized. Note that while off-policy methods (Eyuboglu et al., 2025; Wang et al., 2024; Cao et al., 2025; Snell et al., 2022) only need to generate labels for a single turn, the on-policy methods (Ye et al., 2026; Zhang et al., 2026) roll out responses online and compute logits conditioned on the context, making teacher-side cache reuse even more critical.

4.6 Generalization and Ablation

We further investigate the generality of the proposed method. Table 3 reports the performance of different distillation variants with and without caching, with detailed distillation settings provided in Section D. Overall, the results suggest that the proposed cache-sharing is not specific to any single distillation objective. Across NarrativeQA and SQuAD, cache-sharing distillation largely preserves the average performance of the corresponding non-caching variants, and in some cases even yields improvements under certain objectives. These findings suggest that our proposed cache-sharing distillation can be seamlessly combined with diverse distillation algorithms while maintaining effectiveness relative to standard non-cached distillation. We further conduct experiment on Llama3.1-8B (Grattafiori et al., 2024) in Section G. We also discuss the cache-sharing distillation, internal routing and Self-Gating threshold with empirical ablation study in Section F.

5 Conclusion

We formulate context distillation as latent memory management. Instead of accumulating all contexts into a single parameter state, we distill each context into an independent LoRA adapter and manage these adapters through retrieval and Self-Gating. This design enables modular storage, query-aware memory selection, and safe fallback to the base model when latent memory is unnecessary. Experiments show that our method outperforms cumulative distillation baselines in realistic non-oracle settings and preserves general model capabilities on context-agnostic queries, with a lightweight management system.

Limitations

Although context distillation with LoRA as latent memory can reduce inference cost, it currently still exhibits a performance gap compared with ICL under the oracle setting. Future work may close this gap by developing more effective distillation strategies, such as on-policy distillation methods (Ye et al., 2026; Zhang et al., 2026).

Furthermore, storage overhead remains a limitation, as saving a separate adapter can still be costly. Future work may investigate more parameter-efficient modules for context distillation, such as BitFit (Zaken et al., 2022), to further reduce the memory footprint.

References

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

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

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

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

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

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and 1 others. 2025. Cartridges: Lightweight and general-purpose long context representations via self-study. *arXiv preprint arXiv:2506.06266*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and 1 others. 2026. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, and 1 others. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

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

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

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

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

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

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173.

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

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

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

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

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. 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)*, pages 4149–4158.

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

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

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *arXiv e-prints*, pages arXiv–2412.

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

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9.

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

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, and 1 others. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *International conference on machine learning*, pages 12697–12706.

A Cumulative Context Distillation

Given a stream of contextual data $\mathcal{C} = \{c_1, c_2, \ldots, c_t\}$, existing methods such as Infinite-ICL (Cao et al., 2025) and TempLora (Wang et al., 2024) update the model's latent knowledge following a cumulative distillation paradigm. Specifically, at step i, the model parameters θ_{i-1} are updated to θ_i by minimizing the Kullback-Leibler (KL) divergence between the updated model and the previous model conditioned on the new context:

$$\theta_i = \arg\min_{\theta} D_{KL}\big(\pi_{\theta_{i-1}}(\cdot|q, c_i) \parallel \pi_\theta(\cdot|q)\big) \tag{12}$$

where q is the query.

To understand the theoretical limit of this paradigm, let us consider an ideal scenario. Assume that for all steps i, there exists an optimal set of parameters θ_i^* such that the unconditioned output distribution perfectly matches the context-conditioned distribution from the previous step:

$$\pi_{\theta_i^*}(y|q) = \pi_{\theta_{i-1}^*}(y|q, c_i), \forall q \tag{13}$$

with the initial state defined as:

$$\pi_{\theta_1^*}(y|q) = \pi_\theta(y|q, c_1), \forall q \tag{14}$$

where θ denotes the parameters of the base model. Note that prepending context c_i to query q can be conceptually viewed as forming an augmented query $q_i = [c_i, q]$. As illustrated in Figure 10, by recursively applying Eq. 13 and Eq. 14, we can derive the equivalent distribution at step i:

$$\pi_{\theta_i^*}(y|q) = \pi_\theta(y|q, [c_1, c_2, \ldots, c_i]), \forall q \tag{15}$$

Equation 15 reveals a critical property: under this idealized recursive teacher assumption, the target distribution coincides with the base model conditioned on the full concatenated context.

Consequently, this cumulative paradigm suffers from several inherent limitations:

- **Inherited Long-Context Degradation:** Since the theoretical limit relies on the base model processing $[c_1, \ldots, c_i]$, the distilled model may inherit the base model's weaknesses regarding long context.
- **Noise Assimilation:** In real-world streaming data, not all contexts are beneficial for answering subsequent queries. The standard cumulative approach indiscriminately distills all provided tokens, forcing the model to absorb noise and irrelevant information into its latent knowledge.

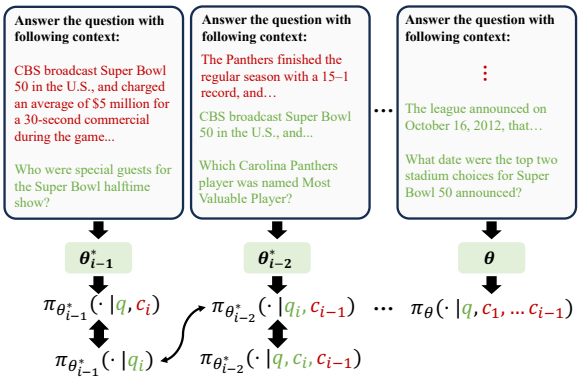


Figure 10: Theoretical upper bound of cumulative distillation paradigm.

B Dataset

B.1 Synthetic Query Generation

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:

Following previous works (Cao et al., 2025; Eyuboglu et al., 2025), we construct synthetic queries from the NarrativeQA and SQuAD corpus. For each document, we use the document summary as the source context and prompt an instruction-tuned causal language model to generate natural-language questions. The prompt requires the model to produce a numbered list of distinct questions that are highly related to the given context, while explicitly excluding answers, answer options, or additional explanatory text. The prompt template is shown in Prompt 1.

For each document, we repeatedly sample from the generator until collecting up to $k = 1000$ valid questions. We use stochastic decoding with temperature sampling to encourage diversity. Since the raw model outputs may contain formatting artifacts,

Table 4: Dataset statistics for NarrativeQA (Kočiskỳ et al., 2018) and SQuAD (Rajpurkar et al., 2016). Document length, queries per document, and query length are reported as mean $\pm$ standard deviation; all lengths are measured in tokens.

Dataset	Split	#Docs	Doc Length	#Queries per Doc	Query Length
NarrativeQA	Train	355	731.44 ± 259.76	1000.00 ± 0.00	13.77 ± 6.88
	Test			29.74 ± 2.45	10.92 ± 3.77
SQuAD	Train	300	158.93 ± 80.08	1000.00 ± 0.00	13.54 ± 5.42
	Test			8.81 ± 4.27	12.71 ± 4.15

we parse the generations with a regular expression that extracts interrogative sentences from numbered or bulleted lists. We then normalize each candidate by removing assistant-style prefixes and truncating the text at the first question mark.

B.2 Dataset Statistics

We report the dataset statistics used in our paper in Table 4. We use the document from the test dataset of NarrativeQA (Kočiskỳ et al., 2018) and validation dataset of SQuAD (Rajpurkar et al., 2016). For train split, we generate queries with the process in Section B.1, and generate corresponding responses with base model during training. For test split, we use the original queries and answers in the test dataset of NarrativeQA and validation dataset of SQuAD.

C Experiment Details

We evaluate in a document-level adaptation setting. At distillation time, the system has access to the target document but not to the benchmark evaluation questions or gold answers. We generate synthetic training queries from the document and teacher responses from the base model, then evaluate on the original benchmark questions. Thus, our setting measures whether a document can be compiled into a reusable latent memory for future queries about that document.

C.1 Hyperparameter

For all context distillation method in our paper, we use the same hyperparameter for training, as shown in table 5.

C.2 Environment

All experiments were conducted on a single GPU. For Qwen2.5-0.5B, we used an NVIDIA L40 GPU, while experiments with Qwen2.5-7B and Llama-3.1-8B were run on an NVIDIA H100 GPU. Each training run was completed within approximately 1–2 days.

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

C.3 Metrics

We evaluate SQuAD with Exact Match (EM) and token-level F1, following the standard SQuAD evaluation protocol. Both predictions and reference answers are normalized prior to scoring by lowercasing, removing punctuation and English articles, and fixing whitespace. For instances with multiple reference answers, we compute the score against each reference and report the maximum score.

For NarrativeQA, we report ROUGE-1 and ROUGE-L, computed using the HuggingFace `evaluate` implementation of ROUGE with its default configuration. Model predictions are compared against the provided gold reference answers.

C.4 Baseline Evaluation

LoRA-only. In the LoRA-only setting, all document-specific information is stored in the LoRA adapter. Given a query q_i associated with document c_i, whose corresponding latent memory

is encoded by the adapter $\Delta\theta_i$, we evaluate the following settings:

- ***Latest***: uses the most recent adapter $\Delta\theta_n$ to answer the query, i.e., generates the response as $f_{\theta+\Delta\theta_n}(q_i)$.
- ***Oracle***: assumes access to the correct adapter $\Delta\theta_i$ associated with document c_i, i.e., generates the response as $f_{\theta+\Delta\theta_i}(q_i)$.
- ***Shift***: evaluates the query q_i using the next adapter $\Delta\theta_{i+1}$, i.e., generates the response as $f_{\theta+\Delta\theta_{i+1}}(q_i)$. This setting is designed to probe whether information from c_i is forgotten or overwritten after updating to the subsequent adapter.

LoRA+Context. InfiniteICL (Cao et al., 2025) treats the LoRA adapter as long-term memory and the input context as short-term memory. Given a query q_i associated with document c_i, with the corresponding latent memory encoded by $\Delta\theta_i$, we evaluate:

- ***Latest***: uses the latest available context c_n together with the preceding adapter $\Delta\theta_{n-1}$, i.e., generates the response as $f_{\theta+\Delta\theta_{n-1}}(q_i, c_n)$.
- ***Oracle***: assumes access to the correct context c_i and the adapter immediately preceding it, $\Delta\theta_{i-1}$, i.e., generates the response as $f_{\theta+\Delta\theta_{i-1}}(q_i, c_i)$.
- ***Shift***: evaluates the query q_i with the subsequent context c_{i+1} and the adapter $\Delta\theta_i$, i.e., generates the response as $f_{\theta+\Delta\theta_i}(q_i, c_{i+1})$.

C.5 Router Training Details

We train a lightweight router to select one adapter from the retrieved top-k candidates for each query. Given a query q, retrieval returns $\mathcal{C}_k(q) = \{c_1, \ldots, c_k\}$ with similarity scores $s(q, c_i)$. A separate router is trained for each dataset, routing mode, and value of k.

For each candidate c_i, we use the feature vector

$$\mathbf{x}_{q,i} = \left[s(q, c_i),\ s(q, c_1) - s(q, c_i),\ H^{(1)}_{q,i},\ \mathbf{h}^{(1)}_{q,i} \right], \tag{16}$$

where $s(q, c_i)$ is the retrieval similarity, $s(q, c_1) - s(q, c_i)$ is the gap to the top-ranked candidate, $H^{(1)}_{q,i}$ is the entropy of the first-token predictive distribution, and $\mathbf{h}^{(1)}_{q,i}$ denotes the first 128 dimensions of the first-token hidden state.

We use candidate-level binary supervision:

$$y_{q,i} = \mathbb{1}[c_i = c^\star(q)], \tag{17}$$

where $c^\star(q)$ is the gold document/adapter id. If $c^\star(q) \notin \mathcal{C}_k(q)$, all candidates for q are labeled as negative. The router is therefore trained as a binary scorer. All candidate features from the training split are flattened and standardized using training-set statistics. The router is a two-layer MLP that outputs one scalar logit per candidate:

$$r_\theta(\mathbf{x}) = \mathrm{MLP}_\theta(\tilde{\mathbf{x}}). \tag{18}$$

We train the router with weighted binary cross-entropy loss, using

$$w_+ = \frac{N_{\text{neg}}}{\max(N_{\text{pos}}, 1)} \tag{19}$$

to compensate for class imbalance, and optimize with AdamW. Unless otherwise specified, we use 100 epochs, learning rate 5×10^{-2}, weight decay 10^{-3}, and full-batch training.

At inference time, the router scores candidates independently and selects

$$\hat{c}(q) = \arg\max_{c_i \in \mathcal{C}_k(q)} r_\theta(\mathbf{x}_{q,i}), \tag{20}$$

which is then used as the adapter id for downstream LoRA generation.

The router is trained only on synthetic QA examples in Section B.1. For each dataset, we first construct a training split from generated questions associated with the adapter pool, and use the corresponding document id as the positive adapter label. In contrast, evaluation is performed on the original real QA split: SQuAD uses real questions from the SQuAD validation split, and NarrativeQA uses real questions from the NarrativeQA test split.

C.6 Gating Threshold Selection

We use the entropy of the first generated token as the gating signal. Given a query–adapter pair, the LoRA-adapted model produces the next-token logits at the first decoding step. We convert the logits into a probability distribution

$$p_0(v) = \mathrm{softmax}(z_0)_v,$$

and compute the first-token entropy

$$\mathcal{H}_0 = -\sum_{v \in \mathcal{V}} p_0(v) \log p_0(v).$$

A lower value of $\mathcal{H}_0$ indicates that the LoRA adapter is more confident for the current query, while a higher value suggests that the query is out of distribution for the selected adapter. Therefore, we accept the LoRA adapter only when

$$\mathcal{H}_0 < \lambda,$$

and otherwise falls back to the base model.

The threshold λ is selected on a held-out threshold-selection split, not on the final test split. Specifically, we use synthetic queries in Section B.1 for NarrativeQA and use the queries in CommonSenseQA training split. In contrast, evaluation is performed on real questions from the NarrativeQA test split and CommonSenseQA validation split. We further analyze the sensitivity to λ in Section F.2.

D Integration with Different Distillation Objectives

While the preceding experiments primarily adopt standard context distillation with forward KL divergence, our training pipeline is compatible with a broader class of distillation objectives. We therefore integrate the proposed prefix caching mechanism with several representative distillation algorithms and examine whether caching affects downstream task performance. Specifically, we consider four commonly used distillation objectives.

Forward KL. Forward KL is widely adopted in prior context distillation methods (Cao et al., 2025; Eyuboglu et al., 2025; Li et al., 2026). It encourages the student distribution without the context to match the teacher distribution conditioned on the full context:

$$D_{\mathrm{KL}}(\pi_\theta(\cdot \mid q, c) \,\|\, \pi_{\theta+\Delta\theta}(\cdot \mid q))\,. \tag{21}$$

Reverse KL. Reverse KL has recently been used in on-policy distillation methods (Shenfeld et al., 2026; Hübotter et al., 2026; Zhang et al., 2026), where distillation is formulated from a reinforcement-learning perspective. The objective is given by

$$D_{\mathrm{KL}}(\pi_{\theta+\Delta\theta}(\cdot \mid q) \,\|\, \pi_\theta(\cdot \mid q, c))\,. \tag{22}$$

Top-k Logits. This objective applies KL divergence only over the top-k tokens with the highest probabilities (Raman et al., 2023; Ye et al., 2026). Following OPCD (Ye et al., 2026), we use

$$D_{\mathrm{KL}}^{\text{top-}k}(\pi_{\theta+\Delta\theta}(\cdot \mid q) \,\|\, \pi_\theta(\cdot \mid q, c))\,. \tag{23}$$

EMA Teacher. In this setting, the teacher is not fixed during training. Instead, its parameters are updated as an exponential moving average of the student parameters:

$$\bar{\theta}_t = \alpha\bar{\theta}_{t-1} + (1-\alpha)\theta_t, \tag{24}$$

where $\alpha \in [0, 1)$ is the EMA decay rate. The distillation objective is then computed using the EMA teacher distribution:

$$D_{\mathrm{KL}}\big(\pi_{\bar{\theta}_t}(\cdot \mid q, c) \,\big\|\, \pi_{\theta_t}(\cdot \mid q)\big)\,. \tag{25}$$

E Efficiency

E.1 training efficiency

We evaluate training efficiency on a fixed NarrativeQA-sampled setting with $N = 1000$ question-answer pairs. The benchmark uses Qwen2.5-0.5B with LoRA fine-tuning, bfloat16 precision, batch size 16, and 5 warm-up steps. We evaluate context lengths $C \in \{500, 1000, 2000, 4000\}$ and normalize each answer to 32 tokens.

Each measured step performs a full optimizer update using a temperature-scaled KL loss between teacher and student logits on the answer tokens. Runtime includes both one-time KV precomputation and the measured training loop:

$$t_{\text{total}} = t_{\text{precompute}} + t_{\text{train}}.$$

Warm-up steps are excluded. Peak memory is measured with CUDA peak memory statistics. FLOPs are estimated with DeepSpeed `FlopsProfiler` by profiling one batch, scaling to 1000 queries.

E.2 Inference efficiency

We evaluate inference computation on NarrativeQA with Qwen2.5-0.5B under fixed-length generation. Each run uses 100 sampled queries and generates 16 tokens per query. We report total FLOPs. FLOPs are measured with DeepSpeed `FlopsProfiler`. The reported total is

$$\mathrm{FLOPs}_{\text{total}} = \mathrm{FLOPs}_{\text{routing}} + \mathrm{FLOPs}_{\text{generation}},$$

where cached variants decompose routing/generation into prefix-prefill and cache-reuse terms.

F Ablation Study

F.1 Ablation on Internal Routing

We ablate the effect of internal routing by varying the number of retrieved adapters available to

Table 6: Performance comparison under different retrieval settings on NarrativeQA. Best results within each model are highlighted in bold.

Model	Setting	Acc.	R-1	R-L
Qwen 2.5 0.5B	w/o internal routing	65.10%	23.47	22.54
	Retrieval@3	67.30%	**24.54**	**23.57**
	Retrieval@5	**67.50%**	24.44	23.46
Qwen 2.5 7B	w/o internal routing	64.97%	27.18	26.41
	Retrieval@3	67.59%	28.46	27.68
	Retrieval@5	**67.94%**	**28.64**	**27.86**

Table 7: Performance comparison under different retrieval settings on SQuAD. Best results within each model are highlighted in bold.

Model	Setting	Acc.	EM	F1
Qwen 2.5 0.5B	w/o internal routing	73.70%	26.73	41.26
	Retrieval@3	**76.90%**	**28.30**	43.11
	Retrieval@5	76.60%	28.14	**43.25**
Qwen 2.5 7B	w/o internal routing	73.81%	34.85	44.94
	Retrieval@3	**76.91%**	**36.34**	**46.57**
	Retrieval@5	76.00%	36.30	46.45

the model. In this setting, *w/o internal routing* serves as the baseline, where the model directly uses a single top-ranked retrieved adapter without further selection. In contrast, *Retrieval@3* and *Retrieval@5* enable internal routing, allowing the model to select an adapter from multiple retrieved candidates. We report both downstream task performance and routing accuracy, where the latter measures whether the routing module selects the appropriate adapter from the retrieved candidate set.

As shown in Tables 6 and 7, enabling internal routing consistently improves performance across both model scales and datasets. For Qwen 2.5 0.5B on NarrativeQA, *Retrieval@3* improves routing accuracy by 2.20, Rouge-1 by 1.07, and Rouge-L by 1.03 over the baseline without internal routing. *Retrieval@5* further improves routing accuracy by 2.40 and achieves the best accuracy for this model, although its Rouge scores are slightly lower than those of *Retrieval@3*. On SQuAD, the same model also benefits from internal routing: *Retrieval@3* improves routing accuracy by 3.20, EM by 1.57, and F1 by 1.85. Increasing the retrieval size to 5 yields the largest F1 gain of 1.99, but leads to slightly smaller improvements in routing accuracy and EM compared with *Retrieval@3*.

Similar trends are observed for Qwen 2.5 7B. On NarrativeQA, *Retrieval@3* improves routing accuracy by 2.62, Rouge-1 by 1.28, and Rouge-L by 1.27. *Retrieval@5* achieves the best overall performance, with gains of 2.97 in routing accuracy, 1.46 in Rouge-1, and 1.45 in Rouge-L. On SQuAD, *Retrieval@3* obtains the best results, improving routing accuracy by 3.10, EM by 1.49, and F1 by 1.63. In contrast, *Retrieval@5* brings smaller gains of 2.19 in routing accuracy, 1.45 in EM, and 1.51 in F1, suggesting that retrieving more adapters does not always lead to better routing decisions.

Overall, these results indicate that internal routing provides a clear benefit over relying on a single retrieved adapter. By routing among multiple retrieved adapters, the model can exploit complementary task-specific capabilities and produce more accurate predictions. The routing accuracy results further support this conclusion: enabling internal routing consistently improves adapter selection accuracy over the baseline, and improvements in routing accuracy generally align with gains in downstream performance. However, the comparison between *Retrieval@3* and *Retrieval@5* also shows that a larger candidate set may introduce additional ambiguity, especially on SQuAD, where *Retrieval@3* already provides sufficient candidates for effective routing. This suggests that internal routing benefits from a balanced retrieval size that provides enough diversity while avoiding excessive noisy candidates.

F.2 Ablation on Self-Gating Threshold

We analyze how the first-token entropy threshold affects the activation behavior of Self-Gating in the hybrid setting. For each evaluation sample, we first retrieve the top-1 LoRA adapter using the same dense retriever as in the main hybrid evaluation. Given the retrieved adapter, we compute the first-token predictive entropy H_i under the LoRA-adapted model, using the same shared-prefix KV-cache computation as in the inference pipeline. Following the Self-Gating mechanism, the system activates the retrieved LoRA adapter when its first-token entropy is below a threshold λ, and otherwise falls back to the base model:

$$g_i(\lambda) = \mathbf{1}\{H_i < \lambda\}, \tag{26}$$

where $g_i(\lambda) = 1$ denotes continuing generation with the retrieved LoRA adapter. In the hybrid evaluation, NarrativeQA queries are context-specific and are expected to benefit from the retrieved latent memory. Therefore, activating the retrieved LoRA

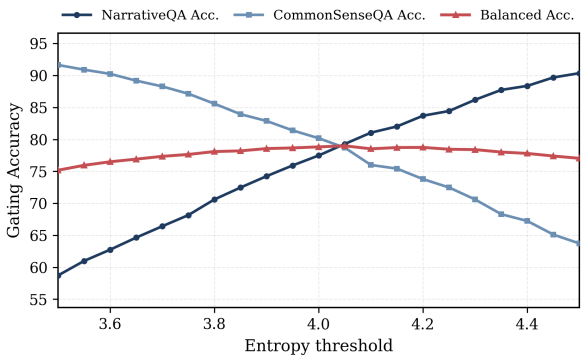


Figure 11: Effect of the first-token entropy threshold on Self-Gating. While the activation and fallback accuracies vary monotonically with the threshold, the balanced accuracy remains stable across a broad operating region, indicating that Self-Gating is insensitive to precise threshold tuning.

adapter is treated as the correct gating decision for NarrativeQA. In contrast, CommonSenseQA queries are context-agnostic and should be answered using the base model without activating an irrelevant latent memory. Thus, falling back to the base model is treated as the correct gating decision for CommonSenseQA. The NarrativeQA gating accuracy is defined as

$$\mathrm{Acc}_{\mathrm{NQA}}(\lambda) = \frac{1}{|\mathcal{D}_{\mathrm{NQA}}|} \sum_{i \in \mathcal{D}_{\mathrm{NQA}}} \mathbf{1}\{H_i < \lambda\}. \tag{27}$$

The CommonSenseQA gating accuracy is defined as

$$\mathrm{Acc}_{\mathrm{CSQA}}(\lambda) = \frac{1}{|\mathcal{D}_{\mathrm{CSQA}}|} \sum_{i \in \mathcal{D}_{\mathrm{CSQA}}} \mathbf{1}\{H_i \geq \lambda\}. \tag{28}$$

We further report the balanced gating accuracy:

$$\mathrm{Acc}_{\mathrm{Balanced}}(\lambda) = \frac{1}{2}\left(\mathrm{Acc}_{\mathrm{NQA}}(\lambda) + \mathrm{Acc}_{\mathrm{CSQA}}(\lambda)\right). \tag{29}$$

We sweep the entropy threshold λ from 3.5 to 4.5 with a step size of 0.05, covering the operating region around $\lambda = 4.0$. Evaluation is conducted on the NarrativeQA test split and the CommonSenseQA validation split. Figure 11 plots the NarrativeQA activation accuracy, CommonSenseQA fallback accuracy, and balanced accuracy as functions of the entropy threshold.

The main observation is that Self-Gating is robust to the precise choice of the entropy threshold. Although changing λ shifts the relative preference between activating the retrieved LoRA adapter and falling back to the base model, the balanced accuracy remains nearly flat around the operating region. In particular, for thresholds from $\lambda = 3.75$ to $\lambda = 4.25$, the balanced accuracy stays within a narrow range from 77.65% to 79.00%. The best value in this sweep is 79.00% at $\lambda = 4.05$, while neighboring thresholds achieve comparable performance.

This insensitivity indicates that Self-Gating does not rely on finely tuned thresholds to distinguish when latent memory should be activated. Instead, the first-token entropy of the LoRA-adapted model provides a stable confidence signal: across a broad interval of λ, the system maintains a similar balance between using context-specific latent memory for NarrativeQA and abstaining from irrelevant memory activation for CommonSenseQA. These results support the practical robustness of the proposed gating mechanism in non-oracle hybrid settings.

F.3 Ablation on Cache-Sharing Distillation

Table 8: Ablation study on Qwen2.5-0.5B evaluated on NarrativeQA and SQuAD.

Setting	NarrativeQA		SQuAD	
	ROUGE-1 ↑	ROUGE-L ↑	EM ↑	F1 ↑
Ours	**31.23**	**30.22**	**33.38**	**50.05**
w/o cache-sharing	26.99	26.00	27.02	43.47

Previous work (Wang et al., 2024) discuss the similar cache sharing mechanism for long text generation, that directly apply KV-Cache after LoRA updates, instead of recomputing them every time. Therefore, we conduct ablation study on whether cache-sharing distillation is important for the inference pipeline in our paper, i.e. we first compute KV-cache with base model, and reuse it with Lora model for generation.

As shown in Table 8, removing cache-sharing distillation consistently degrade performance on both datasets. On NarrativeQA, ROUGE-1 and ROUGE-L drop by 4.24 and 4.22, respectively. On SQuAD, EM and F1 decrease by 6.36 and 6.58. These results indicate that directly reusing KV-caches computed by the base model for the LoRA-adapted model introduces a notable mismatch. This degradation is likely caused by the discrepancy between the base-model cached states and the updated attention computation after applying LoRA. Cache-sharing distillation explicitly exposes the model to this inference setting during training, encourag-

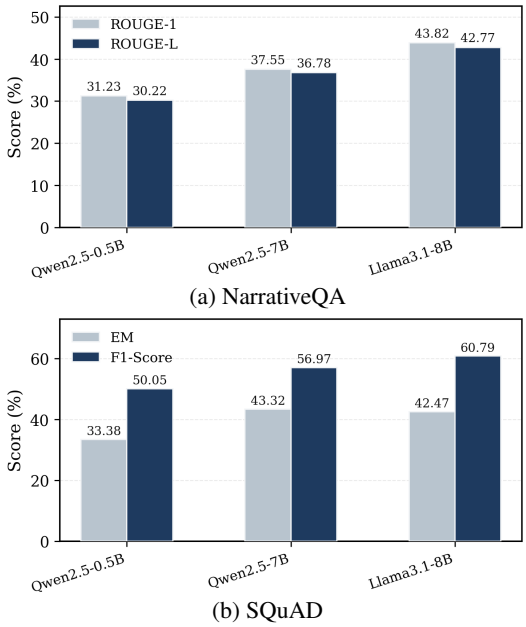


Figure 12: Results across different models.

ing the LoRA model to remain compatible with base-model KV-caches.

G Experiments Across Models

To further examine the robustness and scalability of our method, we evaluate it with different backbone models, including Qwen2.5-0.5B, Qwen2.5-7B, and Llama3.1-8B. The results are shown in Figure 12. Overall, our method consistently achieves strong performance across all evaluated backbones on both NarrativeQA and SQuAD, indicating that the proposed approach is not tied to a specific model family or parameter scale.

These results suggest that our method generalizes well across model scales and architectures. In particular, the consistent gains from smaller to larger backbones indicate that the proposed design can leverage improved model capacity, while still remaining effective even when applied to relatively compact models.