MoNe: Modular Neural Memory for Efficient Long Context Inference

Anonymous authors
Paper under double-blind review

Abstract

Long-context reasoning is essential for applications such as personalized assistants, document QA, and agentic systems, yet standard in-context learning (ICL) becomes inefficient and unreliable as context length grows. We introduce **MoNe** (**Mo**dular **Ne**ural Memory), a lightweight plugin that equips frozen pretrained Transformers with test-time learned fast-weight memories, without modifying or retraining the backbone. MoNe processes long contexts in segments and writes information into these memories through block-localized gradient updates, without backpropagating through the Transformer backbone. During query inference, the adapted memories provide context-dependent residuals at each Transformer layer without including the original context in the model input, while remaining reusable across queries and incrementally extensible as new context arrives. Across RULER, BABILong, SQuAD, and HotpotQA, MoNe generalizes to 128K-token contexts while substantially outperforming the baselines at long context lengths and reducing both peak GPU memory and total FLOPs by approximately 80% relative to ICL at 128K.

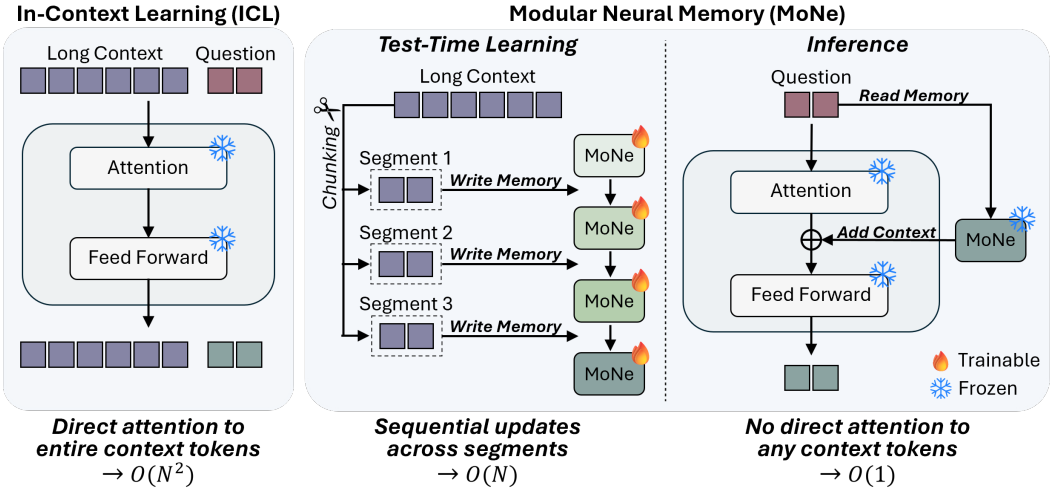


Figure 1: Comparison of ICL and MoNe for long-context processing. ICL provides the full context and query together as input, whereas MoNe writes the context into fast-weight neural memories and answers queries using the memory state, without including the context tokens in the input.

1 Introduction

Long-context processing has become essential to practical AI applications. Personalized assistants must reason over user histories, document QA systems must process lengthy contracts or medical records, and agentic systems must synthesize information spanning tens of thousands of tokens. In this work, we use *in-context learning* (ICL) to refer to the standard long-context setting in which the context and query are provided together as input. However, scaling ICL to long contexts poses challenges in effectiveness and efficiency. As context length increases, language models often exhibit substantial performance degradation and struggle to reliably utilize information distributed over long

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

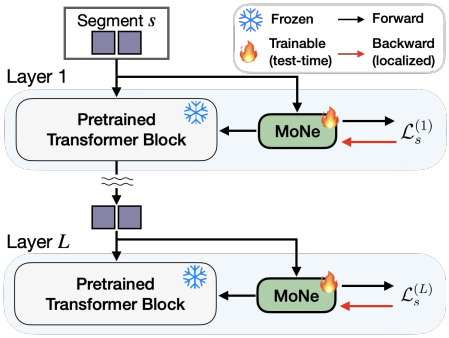


Figure 2: Overview of MoNe's block-localized memory writing at test time.

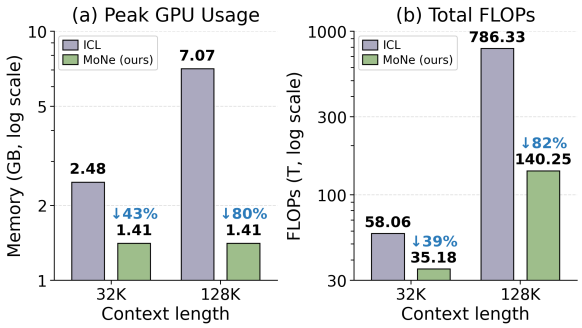


Figure 3: Peak GPU memory and total FLOPs for ICL and MoNe at context lengths of 32K and 128K.

sequences (Liu et al., 2024a; Hsieh et al., 2024). Consequently, the effective context window can be considerably shorter than the maximum sequence length that the model technically supports.

Long-context ICL is also computationally expensive. In standard self-attention, each token interacts with every other token in the sequence, resulting in computational cost that scales quadratically with sequence length, $O(N^2)$ (Vaswani et al., 2017; Tay et al., 2022). Moreover, when answering a query, the model must process the entire context even if only a small fraction is relevant. This leads to substantial redundant computation, particularly when multiple queries are asked over the same context. Such inefficiency becomes especially problematic for on-device deployment, where computational and memory resources are constrained (Liu et al., 2024b; Alizadeh et al., 2024).

As an alternative, PERK (Chen et al., 2026) has explored addressing long-context processing through test-time learning. It encodes the context into LoRA adapter parameters attached to a frozen pretrained Transformer, enabling subsequent queries to be answered using the adapted parameters without including the original long context in the prompt. This provides a modular way to augment an existing pretrained model while keeping the backbone unchanged. However, encoding the context into the LoRA parameters incurs substantial additional computation compared with standard ICL inference, as the adaptation requires multiple gradient-based updates over the context, with backpropagation through the Transformer computation graph during each update. This motivates a more efficient approach to test-time context encoding that retains compatibility with existing pretrained Transformers while substantially reducing the computational cost of adaptation.

To address this challenge, we propose **MoNe** (**Mo**dular **Ne**ural Memory), a lightweight neural-memory plugin that preserves compatibility with a frozen pretrained Transformer while enabling efficient test-time construction of long-context memory. As illustrated in Fig. 1, MoNe sequentially reads the context in fixed-size segments and updates the fast weights of a neural memory associated with each Transformer block, allowing the memories to accumulate information from the long context without modifying any backbone parameters. At inference time, the query is passed through the adapted neural memory at each block, which produces a *context residual*. This residual is incorporated into the frozen attention branch before its output projection, supplying information encoded from the long context without requiring the query to directly attend over the entire long context, as in ICL.

A key advantage of MoNe is its efficiency in both memory writing and query inference. As shown in Fig. 2, each neural memory is updated through *block-localized gradient updates*, where the fast-weight update for each Transformer block is computed from that block's forward-pass activations using an *associative memory loss*. Because the corresponding Transformer activations are treated as fixed inputs to the inner update, memory writing requires no gradient propagation through the Transformer backbone or across Transformer blocks. Figure 3 quantifies this advantage by measuring peak GPU memory and total FLOPs over the complete pipeline, including both test-time learning and inference. Even when accounting for both phases, MoNe requires substantially less memory and computation than ICL, reducing both by approximately 80% at a context length of 128K. Moreover, once the context has been written into the fast weights, the resulting context-specific fast-weight state can be reused across multiple queries and incrementally extended as new context arrives, without reprocessing previously seen content. Across extensive experiments on RULER, BABILong, SQuAD, and HotpotQA, MoNe not only improves efficiency but also maintains strong long-context performance, generalizing to contexts of up to 128K tokens while outperforming competing baselines.

Our contributions can be summarized as follows:

- We introduce **MoNe**, a modular neural memory that can be attached to a frozen pretrained Transformer to enable long-context reasoning through fast weights learned at test time, without modifying the pretrained backbone.
- We develop an efficient memory-writing mechanism based on *block-localized gradient updates*, in which each neural memory is efficiently updated from the forward-pass activations of its corresponding Transformer block using an associative memory loss. This avoids gradient propagation throughout the entire backbone during test-time learning while enabling reusable and incrementally extensible context-specific fast-weight states.
- We demonstrate strong long-context performance and efficiency across RULER, BABILong, SQuAD, and HotpotQA. MoNe generalizes to contexts of up to 128K tokens while outperforming competing baselines and reducing both peak GPU memory and total FLOPs by approximately 80% relative to ICL at 128K context length.

2 Related Work

Long-Context Problem. Reasoning over long contexts remains challenging for LLMs because self-attention scales quadratically with sequence length and models often fail to robustly exploit information distributed across long inputs. In standard in-context learning (ICL), the long context and query are provided together as model input, yet models can miss facts buried in documents or fail to preserve information from distant positions (Liu et al., 2024a; Hsieh et al., 2024), with performance sometimes further degrading as additional context is provided (Hong et al., 2025). Prior work has sought to address these limitations through retrieval (Lewis et al., 2020; Guu et al., 2020) and context-window extension (Peng et al., 2024; An et al., 2024). RAG retrieves a relevant subset of the available context (Izacard & Grave, 2021), but independently chunked retrieval can struggle when answering a query requires synthesizing fragmented or interdependent information (Trivedi et al., 2023). YaRN (Peng et al., 2024) and DCA (An et al., 2024) instead extend the usable context of pretrained LLMs through positional scaling and training-free chunk-based attention, respectively. Despite these different approaches, long-context information is still represented primarily as textual input during query inference. MoNe instead writes the context into lightweight neural memories at test time, allowing subsequent queries to be answered without retaining the original context as model input.

Test-Time Learning with Neural Memory. Test-Time Training (TTT) (Sun et al., 2025), Titans (Behrouz et al., 2024), and LaCT (Zhang et al., 2026) use inference-time-updated *fast-weight* neural memories to store information from past context without repeatedly attending over the full history. However, these memory mechanisms are integrated into the sequence-modeling architecture and require models to be trained with them in place, limiting their direct applicability to existing pretrained Transformers. More recent work instead augments pretrained models at test time: δ-mem (Lei et al., 2026) attaches an online associative memory to a frozen full-attention backbone while continuing to process the textual context through the backbone, whereas PERK (Chen et al., 2026) encodes the context into LoRA adapter parameters so that subsequent queries can be answered without the original context. PERK's adaptation, however, optimizes a next-token prediction objective through multiple epochs over the context with backpropagation through the Transformer backbone, resulting in substantial test-time encoding cost. In contrast, MoNe writes the context into dedicated neural memory modules using an associative memory loss and block-localized gradient updates, avoiding gradient propagation through the Transformer backbone during test-time learning.

3 Method

3.1 Overview

We consider a pretrained L-layer Transformer with hidden dimension d and backbone parameters $\mathbf{\Phi}$. MoNe augments each Transformer block l with a lightweight neural memory $\mathcal{M}(\cdot; \mathbf{W}^{(l)})$, where $\mathbf{W}^{(l)}$ denotes context-specific *fast weights*. The Transformer backbone remains frozen throughout.

Learning is separated into two stages: *meta-training* before deployment and *test-time learning and inference* after deployment. During meta-training, MoNe learns the meta-parameters $\boldsymbol{\theta}$ that govern

the initialization, writing, updating, and readout of the neural memories. After deployment, both $\mathbf{\Phi}$ and $\boldsymbol{\theta}$ are fixed, and only the instantiated fast weights $\mathbf{W}^{(l)}$ are adapted to each new context.

At test time, the context is processed sequentially in fixed-size segments to construct context-specific fast-weight states through block-localized memory updates. Once the context has been encoded, subsequent queries are processed without including the original context tokens in the model input; instead, the adapted neural memories provide the stored context information during query inference.

3.2 Neural Memory Parameterization

Following Zhang et al. (2026), we parameterize each neural memory as a SwiGLU MLP:

$$\mathcal{M}(\mathbf{x}; \mathbf{W}^{(l)}) = \left(\mathrm{SiLU}(\mathbf{x}\mathbf{W}^{(l)}_{\mathrm{gate}}) \odot \mathbf{x}\mathbf{W}^{(l)}_{\mathrm{in}}\right)\mathbf{W}^{(l)}_{\mathrm{out}}, \tag{1}$$

where

$$\mathbf{W}^{(l)} = \{\mathbf{W}^{(l)}_{\mathrm{in}}, \mathbf{W}^{(l)}_{\mathrm{gate}}, \mathbf{W}^{(l)}_{\mathrm{out}}\} \tag{2}$$

denotes the fast-weight state of the neural memory associated with Transformer block l.

The initial fast weights $\{\mathbf{W}^{(l)}_0\}_{l=1}^{L}$ are shared across examples and learned as part of the meta-parameters $\boldsymbol{\theta}$. For each new context, these initial states are instantiated and sequentially updated during test-time learning, yielding context-specific fast weights $\{\mathbf{W}^{(l)}_S\}_{l=1}^{L}$.

3.3 Block-Localized Memory Writing

Given a long context $\mathbf{C} = (t_1, \ldots, t_N)$, we partition it into non-overlapping segments $\mathbf{s}_1, \ldots, \mathbf{s}_S$ of T tokens and process them sequentially.

Let $\mathbf{x}^{(l)}_{s,j}$ denote the hidden representation of token j in segment s at Transformer block l. We obtain the corresponding memory key and value through layer-specific memory feature maps:

$$\mathbf{k}^{(l)}_{s,j} = f^{(l)}_k\left(\mathbf{x}^{(l)}_{s,j}\right), \qquad \mathbf{v}^{(l)}_{s,j} = f^{(l)}_v\left(\mathbf{x}^{(l)}_{s,j}\right). \tag{3}$$

To write segment s into memory, we define an *associative memory loss* that encourages the neural memory to map each key to its corresponding value:

$$\ell^{(l)}_{s,j}(\mathbf{W}^{(l)}) = -\left\langle \mathbf{v}^{(l)}_{s,j}, \mathcal{M}\left(\mathbf{k}^{(l)}_{s,j}; \mathbf{W}^{(l)}\right)\right\rangle. \tag{4}$$

For segment s, we compute token-local associative gradients with respect to the current fast weights:

$$\mathbf{g}^{(l)}_{s,j} = \nabla_{\mathbf{W}^{(l)}} \ell^{(l)}_{s,j}\left(\mathbf{W}^{(l)}_{s-1}\right), \qquad j = 1, \ldots, T. \tag{5}$$

The inner gradients are taken only with respect to $\mathbf{W}^{(l)}$, while the corresponding Transformer activations and meta-learned MoNe parameters are treated as fixed inputs to the update.

A meta-learned update rule aggregates the token-local gradients conditioned on the current segment activations:

$$\boldsymbol{\mu}^{(l)}_s = \mathcal{U}^{(l)}\left(\left\{\mathbf{g}^{(l)}_{s,j}\right\}_{j=1}^{T}, \mathbf{X}^{(l)}_s\right), \tag{6}$$

where $\mathcal{U}^{(l)}$ denotes the meta-learned update mechanism and $\mathbf{X}^{(l)}_s$ collects the block-l activations of segment s. The fast weights are then updated as

$$\mathbf{W}^{(l)}_s = \mathbf{W}^{(l)}_{s-1} - \boldsymbol{\mu}^{(l)}_s. \tag{7}$$

To maintain a comparable overall writing magnitude across context lengths, we scale the test-time inner learning rate inversely with the number of context segments S. This prevents the cumulative fast-weight update from growing proportionally with context length as more segments are written to memory. The same scaling rule is applied across all evaluated context lengths, and implementation details and base learning rates are provided in Appendix C.

Applying this update sequentially over all context segments yields $\mathbf{W}^{(l)}_S$, the final context-specific fast-weight state. Because the inner updates are localized to the corresponding neural memory, memory writing requires no gradient propagation through the Transformer backbone or across Transformer blocks. Additional implementation details of the update mechanism are provided in Appendix A.

3.4 MEMORY-AUGMENTED QUERY INFERENCE

After all context segments have been processed, the final fast weights $\{\mathbf{W}_S^{(l)}\}_{l=1}^L$ constitute the context-specific fast-weight state. The original context tokens are no longer included in the model input, and the query is processed using the frozen Transformer with these adapted fast weights.

At each Transformer block, the query representation is transformed by the corresponding memory feature map. For a query-token representation $\mathbf{x}_j^{(l)}$, the memory readout is

$$\mathbf{h}_j^{(l)} = \mathcal{M}\left(f_q^{(l)}\left(\mathbf{x}_j^{(l)}\right); \mathbf{W}_S^{(l)}\right). \tag{8}$$

The resulting memory output is passed through a lightweight learned readout transformation, yielding $\widetilde{\mathbf{h}}_j^{(l)}$. We provide the detailed form of this transformation in Appendix A.

The transformed memory output is incorporated as an additive residual to the attention representation before the frozen attention output projection:

$$\widetilde{\mathbf{a}}_j^{(l)} = \mathbf{a}_j^{(l)} + \widetilde{\mathbf{h}}_j^{(l)}, \tag{9}$$

where $\mathbf{a}_j^{(l)}$ denotes the frozen self-attention output for token j before its output projection. The augmented representation $\widetilde{\mathbf{a}}_j^{(l)}$ then follows the original Transformer attention pathway. Thus, the memory residual supplies context information required for query inference without requiring the query to directly attend over the original context tokens.

3.5 META-TRAINING OBJECTIVE

Meta-training optimizes the MoNe meta-parameters $\boldsymbol{\theta}$ so that the block-localized fast-weight updates produce context-specific states that are useful for downstream query answering.

For each training example, the context is first encoded using the updates in Eqs. 5–7, producing the final context-specific states $\{\mathbf{W}_S^{(l)}\}_{l=1}^L$. The query and target sequence are then processed using these adapted fast weights, and the meta-parameters are optimized with the autoregressive outer objective

$$\mathcal{L}_{\text{outer}}(\boldsymbol{\theta}) = -\frac{1}{|\mathbf{y}|}\sum_{r=1}^{|\mathbf{y}|}\log p_{\boldsymbol{\Phi},\boldsymbol{\theta}}\left(y_r \mid \mathbf{q}, \mathbf{y}_{<r}, \{\mathbf{W}_S^{(l)}\}_{l=1}^L\right). \tag{10}$$

Block localization applies only to the inner fast-weight updates used for memory writing. During meta-training, the outer objective is differentiated through these inner updates, allowing the shared fast-weight initialization and the mechanisms governing memory writing, updating, and readout to be learned jointly. The pretrained Transformer parameters $\boldsymbol{\Phi}$ remain fixed throughout meta-training.

4 EXPERIMENTS

In this section, we first detail the experimental setup and then present quantitative results, a detailed computational cost analysis, a comparison with long-context ICL enhancements, and ablation studies.

4.1 EXPERIMENTAL SETUP

Datasets. We evaluate on three tasks from the RULER benchmark (Hsieh et al., 2024), each designed to require precise retrieval or aggregation over the full context: S-NIAH (Single Needle-in-a-Haystack), MK-NIAH (Multi-Key Needle-in-a-Haystack), and FWE (Frequent Words Extraction). We further evaluate on QA1, QA2, and QA3 from BABILong (Kuratov et al., 2024), which require reasoning over one, two, and three supporting facts distributed throughout a long context, respectively. For real-world question answering, we use SQuAD (Rajpurkar et al., 2016) and HotpotQA (Yang et al., 2018) to evaluate single-hop and multi-hop reasoning, respectively. MoNe is meta-trained separately for each task using examples constructed from the corresponding dataset, with training contexts ranging from 1K to 4K tokens. Further details are provided in Appendices B and C.

Table 1: Performance of ICL, PERK, and MoNe (ours) on RULER S-NIAH, MK-NIAH, and FWE across differenet context lengths. Results are grouped by whether the input falls within or beyond the backbone's native 32K context window. * indicates a statistically significant difference between MoNe and ICL ($p < 0.001$).

		Within model context length				*Beyond model context length*			
Task	**Method**	4K	8K	16K	32K	48K	64K	96K	128K
S-NIAH	ICL	0.92	0.97	0.89	0.96	0.67	0.56	0.40	0.22
	PERK	0.99	1.00	0.98	0.77	0.32	0.09	0.02	0.01
	MoNe (ours)	**1.00***	**1.00**	**1.00***	**1.00**	**1.00***	**1.00***	**1.00***	**1.00***
MK-NIAH	ICL	0.83	0.77	0.86	0.81	0.40	0.18	0.10	0.02
	PERK	1.00	1.00	0.99	0.58	0.08	0.01	0.00	0.00
	MoNe (ours)	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***	**0.99***	**1.00***	**0.99***
FWE	ICL	0.58	0.55	0.43	0.44	0.27	0.31	0.30	0.23
	PERK	0.99	0.98	0.99	0.95	0.99	0.98	0.99	0.98
	MoNe (ours)	**1.00***	**1.00***	**0.99***	**1.00***	**1.00***	**1.00***	**1.00***	**1.00***

Table 2: Performance of ICL, PERK, and MoNe (ours) on BABILong QA1, QA2, and QA3 tasks across different context lengths. Results are grouped by whether the input falls within or beyond the backbone's native 32K context window. * indicates a statistically significant difference between MoNe and ICL ($p < 0.001$).

		Within model context length				*Beyond model context length*			
Task	**Method**	4K	8K	16K	32K	48K	64K	96K	128K
QA1	ICL	0.44	0.44	0.41	0.32	0.16	0.07	0.02	0.11
	PERK	1.00	0.98	0.97	0.78	0.63	0.44	0.37	0.25
	MoNe (ours)	**0.99***	**1.00***	**0.99***	**0.98***	**0.98***	**1.00***	**0.98***	**0.96***
QA2	ICL	0.21	0.13	0.12	0.06	0.05	0.03	0.02	0.02
	PERK	0.61	0.56	0.46	0.35	0.37	0.28	0.24	0.27
	MoNe (ours)	**0.68***	**0.66***	**0.72***	**0.73***	**0.74***	**0.70***	**0.72***	**0.63***
QA3	ICL	0.13	0.11	0.07	0.08	0.04	0.05	0.03	0.01
	PERK	0.70	0.60	0.53	0.45	0.42	0.34	0.28	0.25
	MoNe (ours)	**0.77***	**0.75***	**0.68***	**0.73***	**0.73***	**0.65***	**0.68***	**0.70***

Table 3: Performance of ICL, PERK, and MoNe (ours) on SQuAD (single-hop) and HotpotQA (multi-hop) across different context lengths. Results are grouped by whether the input falls within or beyond the backbone's native 32K context window. * indicates a statistically significant difference between MoNe and ICL ($p < 0.001$).

		Within model context length				*Beyond model context length*			
Task	**Method**	4K	8K	16K	32K	48K	64K	96K	128K
SQuAD	ICL	0.56	0.56	0.54	0.45	0.19	0.13	0.07	0.04
	PERK	0.47	0.37	0.23	0.11	0.09	0.08	0.07	0.07
	MoNe (ours)	**0.62**	**0.65**	**0.60**	**0.62***	**0.59***	**0.57***	**0.59***	**0.47***
HotpotQA	ICL	0.26	0.31	0.21	0.19	0.16	0.11	0.07	0.07
	PERK	0.52	0.51	0.51	0.40	0.32	0.26	0.20	0.19
	MoNe (ours)	**0.62***	**0.59***	**0.60***	**0.53***	**0.46***	**0.43***	**0.38***	**0.31***

Table 4: End-to-end computational cost at 32K and 128K context lengths. Parentheses give the change relative to ICL (↓ reduction, ↑ increase). Sub-rows separate test-time learning from inference; method-level memory is the peak across phases, and method-level FLOPs are summed across phases.

Method	Context length 32K		Context length 128K	
	Peak mem. (GB)	FLOPs (T)	Peak mem. (GB)	FLOPs (T)
ICL	2.48	58.06	7.07	786.33
PERK	6.88 (↑177%)	381.41 (↑557%)	6.88 (↓3%)	1525.13 (↑94%)
⌞ Test-time learning	6.88	381.24	6.88	1524.96
⌞ Inference	2.32	0.17	2.32	0.17
MoNe	**1.41** (↓43%)	**35.18** (↓39%)	**1.41** (↓80%)	**140.25** (↓82%)
⌞ Test-time learning	1.41	35.02	1.41	140.09
⌞ Inference	1.32	0.16	1.32	0.16

Evaluation Metrics. For the RULER tasks, S-NIAH and MK-NIAH are evaluated using substring exact match (Sub-EM), which measures whether the ground-truth value appears as a substring of the model output, while FWE uses variable recall, defined as the fraction of the three target words appearing in the model output. For BABILong, QA1, QA2, and QA3 are all evaluated using Sub-EM. For the real-world QA tasks, SQuAD and HotpotQA are evaluated using an LLM-as-judge. All scores are reported on a scale from 0 to 1. We evaluate all tasks at context lengths of 4K, 8K, 16K, and 32K tokens within the backbone's native context window, and at 48K, 64K, 96K, and 128K tokens beyond the native window. Further evaluation details are provided in Appendix E.

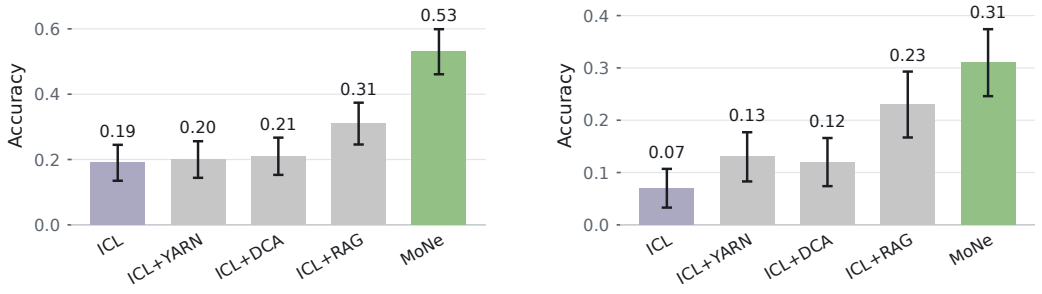
Baselines. We use Qwen2.5-0.5B-Instruct (Qwen et al., 2025) as the frozen backbone for all methods, which supports context lengths up to 32K. ICL feeds the full context together with the query as model input. PERK (Chen et al., 2026) encodes the context into LoRA adapters through test-time gradient updates and subsequently answers queries without the original context. PERK provides the most closely aligned baseline for evaluating long-context reasoning under the context-removal setting. To examine whether established enhancements to ICL can mitigate its long-context degradation, we additionally evaluate ICL with YaRN (Peng et al., 2024), DCA (An et al., 2024), and retrieval-augmented generation (RAG) (Lewis et al., 2020; Guu et al., 2020). YaRN and DCA extend the effective context window, while RAG selects query-relevant portions of the context before inference. For RAG, we split each context into chunks of 256 tokens and use BGE-Large (Xiao et al., 2024) to compute embeddings. The top-5 chunks with the highest cosine similarity to the query are concatenated with the query as model input.

4.2 Evaluation Results

Tables 1–3 report the performance of ICL, PERK, and MoNe across eight tasks and context lengths ranging from 4K to 128K tokens. The results reveal a consistent pattern: MoNe performs strongly across all tasks and maintains substantially more stable performance as the context length increases, including beyond the backbone's native 32K context window.

RULER. Table 1 reports results on S-NIAH, MK-NIAH, and FWE. MoNe maintains near-perfect performance across all tasks and context lengths, achieving scores between 0.99 and 1.00 from 4K to 128K tokens. In contrast, ICL degrades sharply beyond the native context window, reaching 0.22 on S-NIAH, 0.02 on MK-NIAH, and 0.23 on FWE at 128K. PERK performs strongly on S-NIAH and MK-NIAH at shorter context lengths but degrades rapidly as the context grows. At 128K, its performance falls to 0.01 on S-NIAH and 0.00 on MK-NIAH. PERK remains effective on FWE and achieves 0.98 at 128K. MoNe instead maintains stable performance on both retrieval and aggregation tasks throughout the evaluated context lengths.

BABILong. Table 2 reports results on BABILong QA1, QA2, and QA3. MoNe achieves the best performance in nearly all BABILong settings and remains substantially stronger as context length increases. On QA1, MoNe maintains scores between 0.96 and 1.00, while ICL and PERK decrease to 0.11 and 0.25 at 128K, respectively. QA2 and QA3 are more challenging, but MoNe remains stable



(a) 32K context

(b) 128K context

Figure 4: Accuracy of MoNe and long-context ICL enhancements on HotpotQA with 32K and 128K contexts.

as the context length increases. At 128K, MoNe achieves 0.63 on QA2 and 0.70 on QA3, compared with 0.27 and 0.25 for PERK. ICL obtains 0.02 on QA2 and 0.01 on QA3 at the same context length. These results show that MoNe preserves its advantage on tasks that require reasoning over multiple facts distributed throughout the context.

Real-World Question Answering. Table 3 reports results on SQuAD and HotpotQA. MoNe achieves the highest performance at every evaluated context length on both datasets. At 32K, MoNe obtains 0.62 on SQuAD and 0.53 on HotpotQA, compared with 0.45 and 0.19 for ICL and 0.11 and 0.40 for PERK. The advantage remains substantial beyond the native context window. At 128K, MoNe achieves 0.47 on SQuAD, compared with 0.04 for ICL and 0.07 for PERK. On HotpotQA, MoNe achieves 0.31, outperforming ICL and PERK at 0.07 and 0.19, respectively. Although MoNe also degrades at longer context lengths on these tasks, its degradation is substantially smaller than that of either baseline.

4.3 Computational Cost Analysis

Table 4 compares the end-to-end computational cost of the three primary methods, ICL, PERK, and MoNe, including both test-time learning and inference. At 32K tokens, MoNe reduces peak GPU memory and total FLOPs by 43% and 39% relative to ICL, respectively. At 128K tokens, these reductions increase to 80% and 82%, respectively.

MoNe is also substantially more efficient than PERK. At 32K, PERK requires 6.88 GB of peak GPU memory and 381.41 T FLOPs, compared with 1.41 GB and 35.18 T FLOPs for MoNe. At 128K, PERK requires 1525.13 T FLOPs, whereas MoNe requires 140.25 T FLOPs. Across both context lengths, MoNe uses approximately 80% less peak GPU memory and 91% fewer FLOPs than PERK.

For a fixed query and generation length, MoNe's inference cost remains constant across the evaluated context lengths, requiring 1.32 GB of peak GPU memory and 0.16 T FLOPs at both 32K and 128K. Although the FLOPs required for test-time learning increase with context length, its peak GPU memory remains constant at 1.41 GB.

4.4 Comparison with Long-Context ICL Enhancements

Figure 4 compares MoNe with approaches that enhance standard ICL for long-context processing. YaRN and DCA extend the effective context window, while RAG retrieves query-relevant portions of the context before inference. At 32K, YaRN and DCA provide modest improvements over ICL, increasing accuracy from 0.19 to 0.20 and 0.21, respectively. RAG achieves 0.31, while MoNe obtains 0.53. At 128K, ICL decreases to 0.07, while YaRN and DCA achieve 0.13 and 0.12, respectively. RAG reaches 0.23, whereas MoNe obtains 0.31. The results show that these approaches can partially mitigate the long-context degradation of standard ICL, but remain less effective than encoding the context into neural memory.

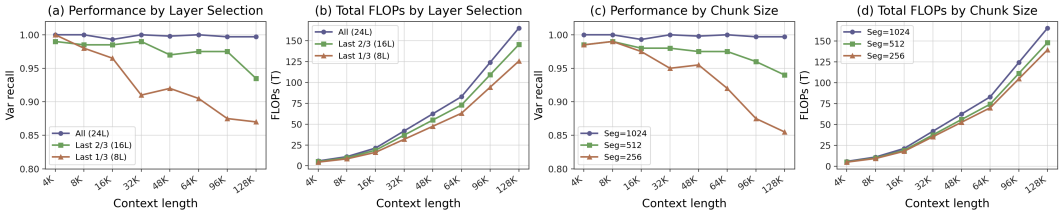


Figure 5: Performance–efficiency trade-offs. Panels (a–b) report variable recall and total FLOPs under different layer coverage; panels (c–d) report the same metrics for different segment sizes.

4.5 Performance-Efficiency Trade-off

Figure 5 presents two ablations on FWE, examining how layer coverage and segment size each contribute to long-context generalization.

Layer Selection. We ablate the effect of attaching MoNe to different subsets of the 24 decoder layers. Figure 5(a) shows that attaching MoNe to all 24 layers maintains variable recall of approximately 1.00 at 4K, 64K, and 128K. Restricting MoNe to the last 16 layers (the final $\frac{2}{3}$) preserves short-context performance but degrades at longer contexts, reaching 0.98 at 64K and 0.94 at 128K. Restricting further to the last 8 layers (the final $\frac{1}{3}$) leads to a larger degradation, reaching 0.90 at 64K and 0.87 at 128K. Figure 5(b) shows that restricting layer coverage yields only modest FLOPs savings, while performance degrades more substantially as context length increases.

Segment Size. We compare three segment sizes $T \in \{256, 512, 1024\}$ across the same context lengths. Increasing the segment size consistently improves long-context performance: at 64K / 128K, T=1024 achieves approximately 1.00 / 1.00, outperforming T=512 (0.98 / 0.94) and T=256 (0.92 / 0.85). Although all configurations are trained on contexts of up to 4K tokens, smaller segments degrade more sharply at longer test contexts. As shown in Fig. 5(d), smaller segments offer only modest FLOPs savings, while Fig. 5(c) shows that this comes at a larger cost to long-context performance.

5 Conclusion

We introduced **MoNe**, a modular neural-memory plugin that equips frozen pretrained Transformers with long-context capabilities by writing context into test-time learned fast weights through block-localized updates, without backpropagating through the backbone or retaining context tokens during query inference. Across RULER, BABILong, SQuAD, and HotpotQA, MoNe generalizes from meta-training contexts of up to 4K tokens to test contexts of up to 128K tokens while maintaining substantially stronger performance as context length increases. At 128K, MoNe reduces peak GPU memory by approximately 80% and total FLOPs by 82% relative to ICL while maintaining strong long-context performance. Together with our ablations showing improved long-context generalization with broader layer coverage and larger segment sizes, these results demonstrate that test-time neural memory provides an effective and efficient alternative to repeatedly processing long textual contexts.

Limitations and Future Work. Our experiments evaluate MoNe using a Qwen2.5-0.5B backbone on long-context retrieval and reasoning tasks spanning both synthetic benchmarks and real-world QA datasets. Extending these results to larger model scales and naturally occurring long conversational workloads (Maharana et al., 2024; Shaham et al., 2023) is an important direction for future work. MoNe's plug-in design does not require modifying or retraining the pretrained backbone parameters when applied to larger backbones, and the efficiency advantages of constant-memory inference may become more pronounced at scale, where KV-cache cost is otherwise extremely prohibitive. Further directions include resource-constrained and continuously adapting settings, such as on-device personalization (Cho et al., 2024; Park et al., 2026) and continual learning (Cho et al., 2023; Dorovatas et al., 2026), as well as parameter-efficient designs (Hu et al., 2022; Hwang et al., 2025) for neural memory to further reduce writing cost and support flexible composition of multiple memory modules.

AI USE STATEMENT

In this work, we used generative AI tools to assist with language editing, proofreading, and improving the clarity and readability of the manuscript. We did not use generative AI tools to generate research ideas, formulate scientific claims, design experiments, analyze results, or draw conclusions. We additionally used generative AI tools to help revise and refine presentation of the text during manuscript preparation. We have reviewed all AI-assisted work to ensure technical correctness and consistency with the reported methodology and findings. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the methodological, implementation, data, and evaluation details necessary to reproduce our experiments in the main paper and appendices. Additional details are provided for the architecture and implementation (Appendix A), dataset construction and splits (Appendix B), training configurations (Appendix C), baseline settings (Appendix D), and evaluation protocols (Appendix E). All reported MoNe results are obtained from a single meta-training run per task using seed 42. For each task, we select the checkpoint with the highest validation score on the in-distribution validation split ($N{=}4$) and use it for subsequent long-context evaluation.

REFERENCES

Keivan Alizadeh, Seyed Iman Mirzadeh, Dmitry Belenko, S. Khatamifard, Minsik Cho, Carlo C. Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. LLM in a flash: Efficient large language model inference with limited memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12562–12584. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.678.

Chenxin An, Fei Huang, Jun Zhang, Shansan Gong, Xipeng Qiu, Chang Zhou, and Lingpeng Kong. Training-free long-context scaling of large language models, 2024. URL https://arxiv.org/abs/2402.17463.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time, 2024. URL https://arxiv.org/abs/2501.00663.

Zeming Chen, Angelika Romanou, Gail Weiss, and Antoine Bosselut. Perk: Long-context reasoning as test-time learning, 2026. URL https://arxiv.org/abs/2507.06415.

Wonguk Cho, Jinha Park, and Taesup Kim. Complementary domain adaptation and generalization for unsupervised continual domain shift learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11442–11452, 2023.

Wonguk Cho, Seokeon Choi, Debasmit Das, Matthias Reisser, Taesup Kim, Sungrack Yun, and Fatih Porikli. Hollowed net for on-device personalization of text-to-image diffusion models. *Advances in Neural Information Processing Systems*, 37:43058–43079, 2024.

Vaggelis Dorovatas, Malte Schwerin, Andrew D. Bagdanov, Lucas Caccia, Antonio Carta, Laurent Charlin, Barbara Hammer, Tyler L. Hayes, Timm Hess, Christopher Kanan, Dhireesha Kudithipudi, Xialei Liu, Vincenzo Lomonaco, Jorge Mendez-Mendez, Darshan Patil, Ameya Prabhu, Elisa Ricci, Tinne Tuytelaars, Gido M. van de Ven, Liyuan Wang, Joost van de Weijer, Jonghyun Choi, Martin Mundt, and Rahaf Aljundi. Position: Modular memory is the key to continual learning agents, 2026. URL https://arxiv.org/abs/2603.01761.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts llm performance. Technical report, Chroma, July 2025. URL https://trychroma.com/research/context-rot.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Junseo Hwang, Wonguk Cho, and Taesup Kim. Pica: Parameter-efficient fine-tuning with column space projection. *arXiv preprint arXiv:2505.20211*, 2025.

Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In Paola Merlo, Jorg Tiedemann, and Reut Tsarfaty (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 874–880, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.74. URL https://aclanthology.org/2021.eacl-main.74/.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 106519–106554. Curran Associates, Inc., 2024. doi: 10.52202/079017-3381. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/c0d62e70dbc659cc9bd44cbcf1cb652f-Paper-Datasets_and_Benchmarks_Track.pdf.

Jingdi Lei, Di Zhang, Junxian Li, Weida Wang, Kaixuan Fan, Xiang Liu, Qihan Liu, Xiaoteng Ma, Baian Chen, and Soujanya Poria. δ-mem: Efficient online memory for large language models. *arXiv preprint arXiv:2605.12357*, 2026.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024a. doi: 10.1162/tacl_a_00638.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32431–32454. PMLR, 2024b.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024.

Sunghyun Park, Jeongho Kim, Hyoungwoo Park, Debasmit Das, Sungrack Yun, Munawar Hayat, Jaegul Choo, Fatih Porikli, and Seokeon Choi. Memory-efficient fine-tuning diffusion transformers via dynamic patch sampling and block skipping. *arXiv preprint arXiv:2603.20755*, 2026.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 31932–31951, 2024.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

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.

Uri Shaham, Maor Ivgi, Avia Efrat, Jonathan Berant, and Omer Levy. Zeroscrolls: A zero-shot benchmark for long text understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 7977–7989, 2023.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=wXfuOj9C7L.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10014–10037, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.557. URL https://aclanthology.org/2023.acl-long.557/.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings, 2024. URL https://arxiv.org/abs/2309.07597.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T. Freeman, and Hao Tan. Test-time training done right. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=Tb9qAxT3xv.

A Architecture and Algorithm Details

Memory feature maps. The memory feature maps used in Secs. 3.3 and 3.4 employ layer-specific query, key, and value projections initialized from the corresponding pretrained attention projections. Because Qwen2.5 uses grouped-query attention, the pretrained key and value projections are expanded by repeating each key–value head across its corresponding query-head group, following the same head expansion used in the pretrained attention module. This produces d-dimensional query, key, and value features, which are then split into H memory heads. The resulting base projections remain frozen, while low-rank adapters are optimized during meta-training. The same feature maps are used consistently for memory writing and reading. The adapter rank and memory dimensions are reported in Table 6.

Multi-head memory and readout. Equation 1 suppresses the memory-head index for readability. Each feature map output is split into H head vectors,

$$f_u^{(l)}(\mathbf{x}) = \mathrm{Concat}_{h=1}^{H} \mathbf{z}_{u,h}^{(l)}, \qquad \mathbf{z}_{u,h}^{(l)} \in \mathbb{R}^{d_h}, \qquad u \in \{q, k, v\},$$

where $Hd_h = d$. Each head applies an independent SwiGLU memory with its own fast weights, and the head outputs are concatenated:

$$\mathcal{M}\left(\mathbf{z}; \mathbf{W}^{(l)}\right) = \mathrm{Concat}_{h=1}^{H} \mathcal{M}_h\left(\mathbf{z}_h; \mathbf{W}^{(l,h)}\right).$$

For query token j, the output of memory head h is

$$\mathbf{h}_{j,h}^{(l)} = \mathcal{M}_h\left(\mathbf{z}_{q,j,h}^{(l)}; \mathbf{W}_S^{(l,h)}\right).$$

A learned bias-free projection followed by SiLU produces one scalar per memory head:

$$\mathbf{g}^{(l)}\left(\mathbf{x}_j^{(l)}\right) = \mathrm{SiLU}\left(\mathbf{x}_j^{(l)} \mathbf{A}_g^{(l)}\right) \in \mathbb{R}^H, \qquad \mathbf{A}_g^{(l)} \in \mathbb{R}^{d \times H}.$$

The h-th scalar is broadcast over the d_h channels of the corresponding memory head. The readout is then

$$\widetilde{\mathbf{h}}_j^{(l)} = \mathrm{Concat}_{h=1}^{H}\left[g_h^{(l)}\left(\mathbf{x}_j^{(l)}\right) \mathrm{RMSNorm}\left(\mathbf{h}_{j,h}^{(l)}\right)\right].$$

In our configuration, $H = 4$ and $d_h = 224$, so the concatenated memory readout has dimension $Hd_h = 896 = d$, matching the attention representation before its output projection.

Fast-weight update. For compactness, Eq. 6 suppresses the internal momentum state of $\mathcal{U}^{(l)}$ and its dependence on the current fast weights. For each memory head h and fast-weight matrix $r \in \{\mathrm{gate}, \mathrm{in}, \mathrm{out}\}$, a learned linear map produces a positive token-wise step size

$$\eta_{s,j,h,r}^{(l)} = \rho_S \,\mathrm{softplus}\left(\left[\mathbf{x}_{s,j}^{(l)} \mathbf{A}_\eta^{(l)}\right]_{h,r} + b_\eta\right),$$

where $b_\eta = \mathrm{softplus}^{-1}(\eta_0)$ is a fixed offset and ρ_S denotes the test-time learning-rate scaling described in Appendix C. A second learned bias-free projection, $\mathbf{A}_\beta^{(l)}$, produces a segment-level momentum coefficient for each head:

$$\beta_{s,h}^{(l)} = \frac{1}{T} \sum_{j=1}^{T} \sigma\left(\left[\mathbf{x}_{s,j}^{(l)} \mathbf{A}_\beta^{(l)}\right]_h\right).$$

The internal update state is initialized as $\mathbf{m}_{0,h,r}^{(l)} = \mathbf{0}$ and updated by

$$\mathbf{m}_{s,h,r}^{(l)} = \beta_{s,h}^{(l)} \,\mathrm{stopgrad}\left(\mathbf{m}_{s-1,h,r}^{(l)}\right) + \sum_{j=1}^{T} \eta_{s,j,h,r}^{(l)} \mathbf{g}_{s,j,h,r}^{(l)}.$$

Here, $\mathbf{g}_{s,j,h,r}^{(l)}$ denotes the component of $\mathbf{g}_{s,j}^{(l)}$ corresponding to memory head h and fast-weight matrix r. The fast weights are updated and rescaled as

$$\mathbf{W}_{s,h,r}^{(l)} = \mathcal{R}^{(l)}\left(\mathbf{W}_{s-1,h,r}^{(l)} - \mathbf{m}_{s,h,r}^{(l)}\right),$$

where $\mathcal{R}^{(l)}$ restores the initialized channel-wise norms. The effective update returned by $\mathcal{U}^{(l)}$ in Eq. 6 is

$$\boldsymbol{\mu}_{s,h,r}^{(l)} = \mathbf{W}_{s-1,h,r}^{(l)} - \mathbf{W}_{s,h,r}^{(l)}.$$

Update order. During context encoding, segment s is processed using $\mathbf{W}_{s-1}^{(l)}$, after which both the updated fast weights $\mathbf{W}_s^{(l)}$ and the momentum state $\mathbf{m}_s^{(l)}$ are retained for the next segment. Query and answer tokens read only from the final fast weights without updating either state. Both the final fast weights and momentum state are retained when the memory is subsequently extended with additional context segments.

B Dataset Details

All datasets are pre-tokenized with the Qwen2.5 tokenizer into fixed-length memory-mapped arrays so that each sample's context is exactly $N \times T$ tokens with $T{=}1024$, and the query and answer are stored separately from the context. No sample ever places any part of the question inside the context. All prompts use the raw completion format (no chat template), which is applied identically to MoNe and to every baseline.

Meta-training Data. MoNe is meta-trained separately for each task using examples constructed from the corresponding training split. Meta-training uses only $N \in \{1, 2, 3, 4\}$ segments, corresponding to 1K–4K tokens. All validation and evaluation examples are disjoint from the meta-training data. Dataset-specific construction and sample counts are described below and summarized in Table 5.

B.1 RULER

We use S-NIAH, MK-NIAH, and FWE from RULER (Hsieh et al., 2024), generated with an implementation that follows the benchmark definitions. S-NIAH contains one key–value pair, MK-NIAH contains four scattered pairs and queries one of them, and FWE asks for the three most frequent non-noise synthetic words. For MK-NIAH, a fresh window of the Paul Graham essay corpus is sampled for each example. Exact prompt templates and generation scripts are included with the released code.

B.2 BABILong

We use tasks QA1 (single supporting fact), QA2 (two supporting facts), and QA3 (three supporting facts) from BABILong (Kuratov et al., 2024). Each sample interleaves the bAbI supporting facts and distractor sentences into a background corpus of PG-19 book text, so that the facts needed to answer are scattered across the whole context rather than concentrated in one region. The question and answer are appended after the context in the same completion format used for RULER.

B.3 SQuAD and HotpotQA

For real-world question answering, we build long-context variants of SQuAD (Rajpurkar et al., 2016) (single-hop) and HotpotQA (Yang et al., 2018) (multi-hop). For a target length of $N \times 1024$ tokens, the gold paragraph(s) for a question are placed at a random position among distractor paragraphs drawn from other questions in the same dataset, until the context budget is filled. For HotpotQA, both gold paragraphs required for the multi-hop chain are always present but are placed independently, so the two hops are generally far apart in the context. The model is supervised on the answer span only (`label_mode=ans`), not on the full next-token stream over the context.

B.4 Splits, Sample Counts, and Context Lengths

Table 5 summarizes the data scale. Crucially, meta-training uses only $N \in \{1, 2, 3, 4\}$ segments, i.e. contexts of 1K–4K tokens, while evaluation spans $N \in \{4, 8, 16, 32, 48, 64, 96, 128\}$, i.e. 4K–128K tokens. The 128K evaluation is therefore a $32\times$ extrapolation beyond the longest context seen during meta-training, and a $4\times$ extrapolation beyond the backbone's native 32K window. Training mixes the four values of N uniformly within each batch-forming epoch.

In addition to the evaluation sets, each task has a held-out in-distribution validation split of 100 samples at $N{=}4$ that is used for checkpoint selection, and a separate 100-sample generation-validation split used to monitor free-running generation quality during meta-training. Neither overlaps the evaluation sets.

Table 5: Dataset sizes and evaluated context lengths. Training contexts contain at most 4K tokens ($N \leq 4$ segments of $T=1024$), whereas evaluation extends to 128K tokens ($N=128$). Evaluation sample counts are reported per context length.

Benchmark	**Task**	**Train samples per N**	**Train N**	**Eval samples per N**
RULER	S-NIAH, MK-NIAH, FWE	50,000	$\{1, 2, 3, 4\}$	200
BABILong	QA1, QA2, QA3	50,000	$\{1, 2, 3, 4\}$	200
SQuAD	—	mixed-N corpus	$\{1, 2, 3, 4\}$	200
HotpotQA	—	mixed-N corpus	$\{1, 2, 3, 4\}$	200

C Training Details

We use Qwen2.5-0.5B-Instruct (Qwen et al., 2025) as the frozen backbone for every method. MoNe attaches a SwiGLU fast-weight memory with $H=4$ heads to each of its 24 decoder layers. Table 6 summarizes the architecture and optimization settings. All runs use seed 42 and are trained with data-parallel replication over 4 NVIDIA A100-SXM4-80GB GPUs.

Table 6: Meta-training configuration by benchmark. The outer-loop learning rates η_{outer}, η_{fw}, and η_{lora} apply to the remaining trainable parameters, the fast-weight initialization, and the memory-feature adapters, respectively; η_0 is the base inner-loop learning rate.

Hyperparameter	**RULER**	**BABILong**	**SQuAD**	**HotpotQA**
Segment size T	1024			
Fast-weight heads H	4			
Head dim. $d_{\text{in}} = d_h$	224			
LoRA rank / α	128 / 128			
Base inner LR η_0	2×10^{-3}	2×10^{-3}	2×10^{-3}	4×10^{-3}
η_{outer}	5×10^{-4}			
η_{fw}	2×10^{-3}	2×10^{-3}	2×10^{-3}	1×10^{-3}
η_{lora}	5×10^{-5}	5×10^{-5}	1×10^{-4}	1×10^{-4}
Epochs	1	1	20	20
Optimizer	AdamW			
Warmup / schedule	500 steps / cosine decay			
Batch size	16			
Precision	`bf16` autocast, `fp32` trainable params			
Seed	42			

MoNe is trained on RULER and BABILong for a single epoch over 50K samples per segment count, which is sufficient because their synthetic generators provide effectively unlimited fresh data. Because SQuAD and HotpotQA provide a finite pool of real paragraphs, MoNe is trained on them for up to 20 epochs with a step cap.

Test-time learning rate. At test time, we scale the base inner-loop learning rate η_0 inversely with the number of context segments S. This compensates for the increasing number of sequential memory updates as the context grows and keeps their cumulative update magnitude approximately stable across context lengths. The same deterministic scaling rule is applied to all evaluation context lengths and requires no additional training.

D Baseline Details

All baselines use the same Qwen2.5-0.5B-Instruct backbone, the same evaluation samples, the same prompt format, and the same decoding settings (greedy, 32 new tokens) as MoNe.

ICL. The full context, the question, and the answer prefix are concatenated and fed to the backbone in one forward pass. We apply *no* truncation and no positional-interpolation trick: at 48K–128K tokens the model operates beyond its native 32K window as-is.

PERK. We follow the setup of Chen et al. (2026), encoding the context into LoRA adapters through test-time gradient updates and then answering the query with the context removed. We use 4 inner-loop update iterations and LoRA rank 256. PERK is the most closely aligned baseline to MoNe because it is the only other method evaluated here that operates under the same context-removal setting.

YaRN. We apply YaRN RoPE scaling (Peng et al., 2024) to the frozen backbone at inference time, with the scaling factor set per context length to $(N \times 1024 + 512)/P_{\max}$, where $P_{\max}$ is the backbone's native maximum position. The 512-token headroom accommodates the question and generated answer. Choosing the factor per length rather than fixing one global factor adapts the positional scale to each evaluated length.

DCA. We use Dual Chunk Attention (An et al., 2024) as implemented in the ChunkLlama codebase, leaving the pretraining-length parameter at its default so that the chunk geometry is derived from the backbone's own native window.

RAG. Each context is split into non-overlapping chunks of 256 tokens and embedded with BGE-Large-en-v1.5 (Xiao et al., 2024). The five chunks with the highest cosine similarity to the question are concatenated with the question and fed to the backbone. Retrieval uses cached chunk embeddings, and generation uses vLLM.

E Evaluation Protocol

Every task is evaluated with greedy decoding and a 32-token generation budget, and all scores are reported on a 0–1 scale. Evaluation is run at $N \in \{4, 8, 16, 32, 48, 64, 96, 128\}$ segments, i.e. 4K–128K tokens.

Metrics. S-NIAH and MK-NIAH use substring exact match (Sub-EM): the score is 1 if the ground-truth value appears as a substring of the generated text, and 0 otherwise. FWE uses variable recall: the fraction of the three target words that appear in the output, so partial credit is possible. BABILong uses Sub-EM against the single-token-to-few-token bAbI answer. SQuAD and HotpotQA use the LLM-as-judge metric described next.

E.1 LLM-as-Judge for SQuAD and HotpotQA

We score SQuAD and HotpotQA with an LLM judge because substring matching does not reliably handle paraphrases or coincidental string matches in free-form answers. The judge is Azure OpenAI `gpt-4o` (API version `2024-06-01`) queried at temperature 0 with a fixed system prompt, and returns a binary `judge_em` together with a short reason string. The judge sees the question, the ground-truth answer, and the prediction; it never sees which method produced the prediction, so the same rubric is applied identically to MoNe, ICL, PERK, YaRN, DCA, and RAG.

The fixed rubric assigns 1 only when the prediction's asserted answer is semantically equivalent to the reference. It accepts paraphrases and formatting variants, rejects coincidental substring matches and contradictory answers, and assigns 0 to uncommitted lists of competing answers when a single answer is expected. We release the full prompt, per-sample judgments, and reason strings with the evaluation code.