TITAN-LLAMA: RETROFITTING PRE-TRAINED LLMS WITH EXTERNAL NEURAL MEMORY MODULES

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformers are efficient to pre-train at scale because attention permits parallel computation across sequence positions, but long-context inference requires increasingly large KV caches. Efficient recurrent alternatives reduce the asymptotic complexity of inference, but their sequential dependence limits token-wise parallelism during training, making them harder to scale. Further, the current paradigm of extending long context tasks creates competing notions of retaining information precisely and keeping long-context inference efficient. To address this gap, we introduce Titan-Llama, a post-training method for retrofitting attention-based language models with a hierarchical memory architecture that updates parameters at test-time. Titan-Llama applies a blockwise attention mask and preserves full-fidelity attention over recent tokens while compressing historical context into recurrent, fixed-size neural memory modules (NMMs). Our formulation allows pre-trained attention layers to be replaced without training a new language model from scratch. On Llama-3.2-1B-Instruct, Titan-Llama raises accuracy over segmented attention windows from 23.87% to 40.51% on QuALITY and from 14.50% to 40.75% on LongHealth. Inference benchmarks show more than 20% faster prefill, 37% faster decode at medium context lengths, and 24% lower steady-state decode memory usage in the evaluated configurations. Together, these results demonstrate that incorporating memory architecture mechanisms that utilize test-time training effectively modulates long-context behavior of pre-trained models, offering a practical route to adapting pre-trained transformers for efficient long-context inference.

1 INTRODUCTION

Transformers are efficient to train at scale because attention permits parallel computation across sequence positions. At inference time, however, retaining keys and values for an increasingly long context cause the KV cache to grow. Linear attention and state-space models offer more efficient recurrent inference, but adopting a different architecture generally requires training a new model rather than reusing a pre-trained attention backbone (Katharopoulos et al., 2020; Gu & Dao, 2024). Scaling large language models requires advances in both representational capacity and inference efficiency, which naturally motivates retrofitting the architecture of transformer-based models to better suit them for inference.

The growing KV cache raises the question of how should a pre-trained transformer best retain information from distant tokens without necessarily storing every historical key and value. Titans (Behrouz et al., 2024) represents distant information in an RNN dubbed the neural memory module (NMM), effectively parameterizing historical tokens as fast weight updates to a fixed-size NMM. This motivates our design in using full attention for local windows and representing older tokens in a fixed-size MLP. It is also worth noting that work in KV cache compaction similarly argues that exacting inference tasks do not necessarily require full fidelity on the KV cache (Zhang et al., 2023; Zweiger et al., 2026).

Our work combines these two ideas and develops a method to post-train lightweight, external modules to improve inference efficiency. We partition attention into local and historical contributions, preserve the local contribution, and train a neural memory module (NMM) to provide information from historical context. At test time, a self-supervised key-value reconstruction loss updates the

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

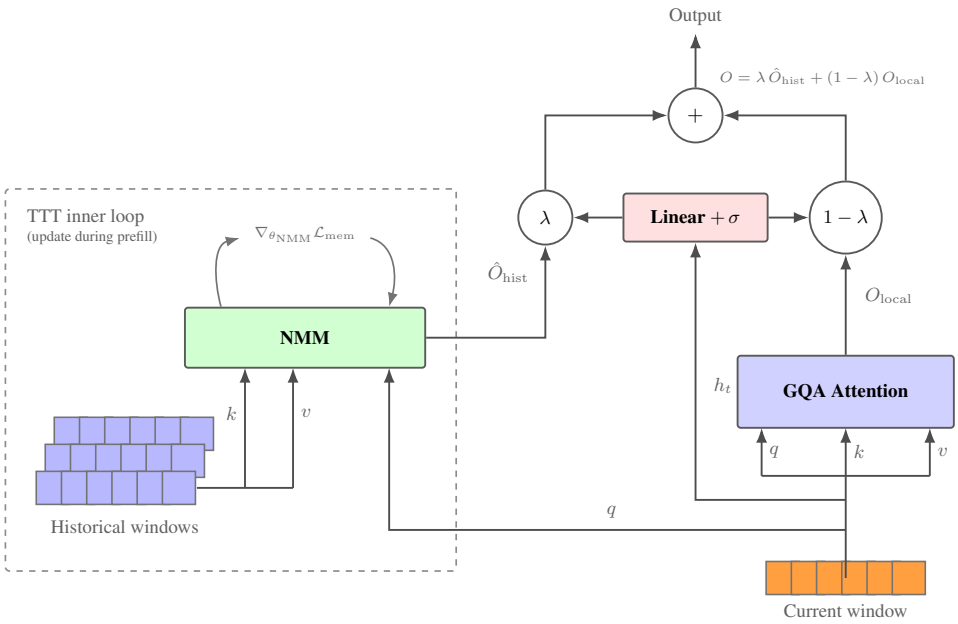

Figure 1: Titan-Llama architecture at a glance. **Left:** The inner loss generates gradients at test-time w.r.t. MLP weights, teaching it to memorize historical attention key-value bindings. **Right:** The query associated with a local hidden state is used to retrieve a value from the NMM. This value, approximating the historical attention output, is then used as a residual and injected into the lambda-gated output stream.

NMM as new chunks arrive. This objective supplies a self-supervised learning signal without labels, although reconstructing key-value pairs does not guarantee exact reproduction of historical attention. Notably, we keep the pre-trained backbone frozen and are able to recover significant performance while training external weights amounting to only 1% of the total parameter footprint.

Two main considerations guide our design:

1. How can we post-train a pre-trained LLM to use Titans as a means to achieve fixed-size context compaction?
2. How can we make the resulting model efficient in practice, accounting for the cost of updating memory during prefill as well as storing and reading it during decode?

2 Related Work

Hierarchical Architectures. Prior work has found success using hierarchical systems to model long dependencies, or more precisely, different mechanisms to model local and distant relationships (Beltagy et al., 2020; Qiu et al., 2020; Kitaev et al., 2020; Bulatov et al., 2022). Many of these approaches, however, do not take advantage of caching and must recompute global attention scores with every new token generated (Patel et al., 2024; Ainslie et al., 2020). Instead, we try to design a method that aligns with the abstraction of prefill and decode phases where historical context can be stored in the NMM via test-time training at prefill time.

Other hierarchical approaches apply downsampling operations to the sequence itself, whereas our time steps follow the chronological arrival of the context. (Hwang et al., 2025; Nawrot et al., 2022; Slagle, 2024).

Retrofitting Pre-trained Models. Modifications to the attention architecture to leverage transformers' training parallelism to fit inference-time objectives have also been studied in prior literature (Bick et al., 2025; Wang et al., 2025a). These approaches differ in how much of the original

model they preserve and how they train the replacement mechanism. For instance, some approaches use distillation objectives to fully convert architectures from scratch while others fine-tune modular components (Eyuboglu et al., 2025; Kojima et al., 2025). Our work aims to fine-tune a small, external set of weights that, through training on next token prediction, can learn how to best modulate a frozen backbone.

Test-time Training. Test-time training (TTT) methods apply gradient updates to a subset of weights at inference time, creating a meta-objective for the slow weights to be able to "learn to learn" (Sun et al., 2020; 2025; Finn et al., 2017). Recent studies utilize TTT approaches to extend the effective context length or to use it to greater effect (Bansal et al., 2025). However, some methods keep the underlying architecture intact, with gradients applying to the MLP layers, decreasing throughput (Feng et al., 2026). The hierarchical approach we use replaces attention for longer-range dependencies with RNNs, which actually improves throughput and average average memory footprint for long-context tasks

The TTT works that provide us the most inspiration and are most similar to ours are Titans (Behrouz et al., 2024) conceptually and LaCT (Zhang et al., 2025) methodologically, but we focus on the post-training dynamics of retrofitting LLMs rather than pre-training or video. Importantly, we also use a self-supervised inner objective rather than final logit prediction, as other TTT approaches do (Tandon et al., 2025; Kuratov et al., 2026; Behrouz et al., 2025).

Titans. To motivate the attention KV binding used in Titans from first principles, consider the vanilla attention operation, which applies a softmax operation to $\mathbf{q}^\top \mathbf{k}$ before applying it to the $\mathbf{v}$ matrix (Sun et al., 2025). In abstract, we can view this as a two-layer MLP where the softmax serves as an activation function and maps keys to values (Han et al., 2026; Wen et al., 2026). In linear attention, by replacing the softmax with a homomorphic function, we collapse the MLP into a single linear map, thereby reducing representational capabilities (Katharopoulos et al., 2020). In the Titans view, efficiency can be achieved by encoding the fast-weight MLP explicitly in neural architecture and simulating fast weights via test-time training (TTT) (Behrouz et al., 2024; 2025).

3 METHOD

3.1 APPROXIMATING HISTORICAL ATTENTION AS A RESIDUAL

As shown in Figure 1, our Titan-Llama layers encode distant information in a fixed-size RNN state and local information with attention, loosely resembling the distinction between lossy long-term memories and vivid recent ones. We now show that contributions to attention from local and historical contexts are easy to partition. This allows us to approximate full attention by predicting the historical output and the mixing weight.

For a query $\mathbf{q}$ at position t and segment length k, let $b_t = k\lfloor t/k \rfloor$ denote the start of the current segment. We partition the available keys and values into historical positions $\mathcal{H} = \{i : i < b_t\}$ and local positions $\mathcal{L} = \{i : b_t \le i \le t\}$. With scores $s_i = \mathbf{q}^\top \mathbf{k}_i / \sqrt{d}$, define

$$Z_{\text{hist}} = \sum_{i \in \mathcal{H}} e^{s_i}, \qquad Z_{\text{local}} = \sum_{i \in \mathcal{L}} e^{s_i},$$

and the attention outputs normalized within each partition:

$$O_{\text{hist}} = \frac{\sum_{i \in \mathcal{H}} e^{s_i} \mathbf{v}_i}{Z_{\text{hist}}}, \qquad O_{\text{local}} = \frac{\sum_{i \in \mathcal{L}} e^{s_i} \mathbf{v}_i}{Z_{\text{local}}}.$$

As in the blockwise formulation of exact attention by Dao et al. (2022), we can partition the softmax numerator and denominator into historical and local contributions, with λ representing the fraction of attention mass assigned to history:

$$O_{\text{attn}} = \frac{\sum_{i \in \mathcal{H}} e^{s_i} \mathbf{v}_i + \sum_{j \in \mathcal{L}} e^{s_j} \mathbf{v}_j}{\sum_{i \in \mathcal{H}} e^{s_i} + \sum_{j \in \mathcal{L}} e^{s_j}}, \qquad \lambda = \frac{Z_{\text{hist}}}{Z_{\text{hist}} + Z_{\text{local}}}.$$

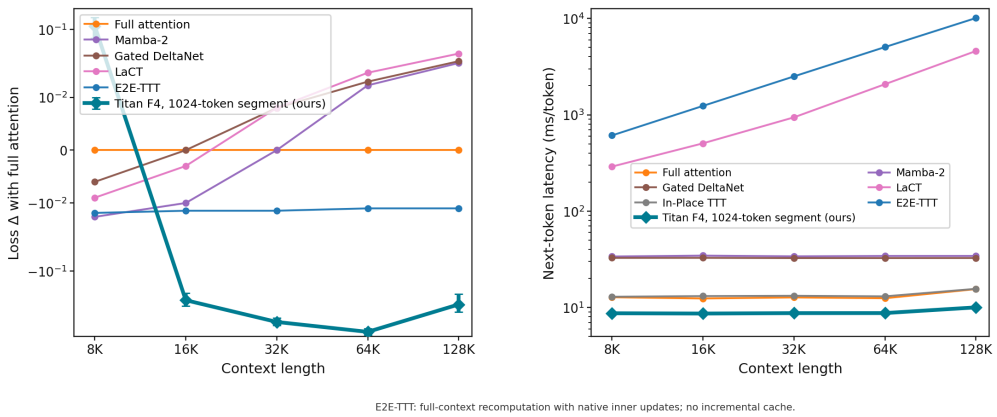

Figure 2: Long-context language modeling and decode latency. **Left.** Change in per-token negative log-likelihood relative to full attention, measured on Pile. Titan-Llama with 1024-token segments beats full attention from 16K through 128K. **Right.** Median single-token decode latency on an A100 at batch size one, after prefill and warmup (lower is better). Titan-Llama's CUDA-graph-enabled decode is faster than the measured full-attention, Mamba-2, Gated DeltaNet, and In-Place TTT implementations across the reported lengths (Dao & Gu, 2024; Yang et al., 2025).

Finally, we can construct the full attention output as a linear interpolation between the historical and local outputs with parameter λ:

$$O = \underbrace{\lambda \operatorname{softmax}\left(\frac{\mathbf{q}^\top \mathbf{K}_{\text{hist}}}{\sqrt{d}}\right)\mathbf{V}_{\text{hist}}}_{\text{historical attention}} + \underbrace{(1-\lambda)\operatorname{softmax}\left(\frac{\mathbf{q}^\top \mathbf{K}_{\text{local}}}{\sqrt{d}}\right)\mathbf{V}_{\text{local}}}_{\text{local attention}} \tag{1}$$

$$\boxed{O = \lambda O_{\text{hist}} + (1-\lambda) O_{\text{local}}.}$$

Our approach replaces the historical term O_{hist} with a read $\hat{O}_{\text{hist}}$ from recurrent memory and predicts its mixing weight. In practice, we predict both λ and O_{hist} and approximate full attention outputs with the expression: $\hat{\lambda}\hat{O}_{\text{hist}} + (1-\hat{\lambda})O_{\text{local}}$. Importantly, this identity does not imply that historical attention can be compressed without loss.

3.2 Reading and Writing at Test-time

Following the derivation in Section 3.1, we seek to approximate historical attention, or $\hat{O}_{\text{hist}}$, with the external neural memory module (NMM) as used in Titans (Behrouz et al., 2024). We make the design choice to blend the retrieved output with a learned gate as a residual added to the local attention output, prior to passing the result to the remaining layer computation. This is similar to other implementations that add the value read from the RNN to the residual stream after attention (Feng et al., 2026) and use it to replace attention entirely in certain layers (Tandon et al., 2025).

We inject the RNN read as a residual into the attention output because this is a natural implementation of equation 1 and aligns with the inner-loss objective described below. This also allows us to modulate an existing LLM backbone without changing any of the parameters already set.

Read. As shown in Figure 1, we use the NMM to memorize the attention key–value mappings from historical attention windows and provide a residual to the current local-attention output, with the two contributions balanced by a per-channel λ vector. More specifically, for each query $\mathbf{q}$, the NMM retrieves a historical attention residual $\hat{\mathbf{O}}_{\text{hist}} = \mathcal{M}(\mathbf{q})$, while a learned gate is computed per channel as:

$$\hat{\lambda} = \sigma\left(\mathbf{W}_\lambda^\top \operatorname{RMSNorm}(\mathbf{q}) + \mathbf{b}_\lambda\right).$$

We combine this local attention output with the NMM retrieval as $\hat{\mathbf{O}}_{\text{full attention}} = \hat{\lambda}\hat{\mathbf{O}}_{\text{hist}} + (1 - \hat{\lambda})\mathbf{O}_{\text{local}}$. We show this expression approximates $\mathbf{O}_{\text{full attention}}$ in Section 3.1. For tokens in a chunk, the NMM read uses the state accumulated from preceding chunks. During inference, once a full chunk has been processed, its keys and values are used to compute the loss that updates the NMM.

Write. To update the NMM at inference time, we minimize a squared-error key–value reconstruction loss over each multi-token chunk c, binding a map between $\mathbf{K}$ and $\mathbf{V}$:

$$\mathcal{L}_c = \|\mathcal{M}(\mathbf{K}_c) - \mathbf{V}_c\|^2 .$$

This loss gives the memory module a training signal at every chunk without requiring labels or a teacher attention output. It encourages the module to retain associations present in the historical keys and values, which future queries may use for retrieval. A future query, however, need not coincide with a stored key, and historical attention can combine values from several positions. Key–value reconstruction is therefore a surrogate for useful memory, rather than an exact objective for reproducing historical attention.

Optimization is done via gradient descent on the NMM once per chunk using learned momentum, decay, and step-size parameters:

$$\mathcal{M}_c = (1 - \alpha_c)\mathcal{M}_{c-1} + \mathbf{S}_c, \qquad \mathbf{S}_c = \eta_c \mathbf{S}_{c-1} - \theta_c \nabla_{\mathbf{M}} \mathcal{L}_c(\mathcal{M}_{c-1}).$$

In practice, we do not perform this update on a per-token scale, and instead use fixed-size token batches to take advantage of modern hardware. For even more efficiency, we use an offline variant of this algorithm by batching the gradient update over multiple chunks.

3.3 Architecture

We make several architectural choices to adapt pretrained LLMs for efficient inference. First, blockwise attention allows us to decouple long-term relationships within the context. This mask complicates the interpretation of equation 1 because earlier layers also use segmented attention, so the historical keys and values differ from those produced by the full-attention model. Blockwise attention also provides a natural threshold for NMM hidden-state updates and inference-time gains from sparse computation.

As shown in equation 1, attention outputs are the mass-weighted sum of local and historical contributions, which we aim to simulate via a predicted λ. This is nontrivial as our use of λ essentially substitutes the usage of a softmax normalization as a means of token reweighting and mixing. We choose to utilize a linear layer on top of the hidden input stream to predict $\hat{\lambda}$ as it is a query-dependent quantity, and we use it to interpolate between O_{local} and $\hat{O}_{\text{hist}}$.

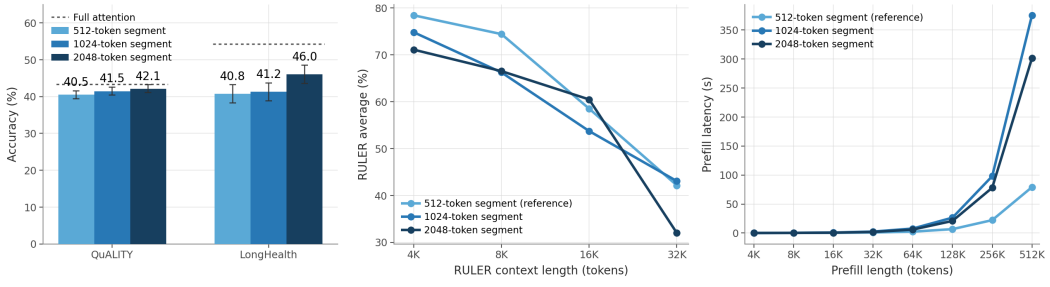

Figure 3: The effects of scaling chunk size on downstream performance and efficiency. **Left.** Scaling chunk size improves performance on long-context reasoning tasks like QuALITY and LongHealth. **Middle.** RULER trends are mixed. Longer chunks help at intermediate lengths but do not consistently improve associative recall. Detailed RULER results can be found in Table 3 in the Appendix. **Right.** Larger chunks increase prefill latency as memory bandwidth saturates.

Table 1: RULER S-NIAH accuracy (%) across context lengths. Titan-Llama matches full attention on 32K pass-key retrieval and achieves the strongest non-full-attention UUID scores at 8K–32K. Relative decode time (↓) averages next-token latency divided by full-attention latency at 8K, 16K, and 32K on A100. † marks timings of publicly available code that may not utilize caching.

	S-NIAH-1 (pass-key)			S-NIAH-2 (number)			S-NIAH-3 (UUID)			Relative decode time ↓
Model	8K	16K	32K	8K	16K	32K	8K	16K	32K	Mean
Full attention	**100.0**	**100.0**	**100.0**	99.0	**100.0**	**100.0**	64.0	64.0	**83.0**	1.00
Mamba-2	99.0	49.0	26.0	99.0	43.0	28.0	77.0	36.0	24.0	2.70
Gated DeltaNet	**100.0**	50.0	26.0	**100.0**	43.0	27.0	91.0	45.0	23.0	2.59
LaCT	98.0	43.0	22.0	**100.0**	43.0	27.0	74.0	34.0	23.0	$45.8^{\dagger}$
TTT-E2E	**100.0**	46.0	24.0	99.0	43.0	28.0	77.0	44.0	24.0	$115^{\dagger}$
Titan-Llama	**100.0**	**100.0**	**100.0**	96.0	78.0	88.0	**92.0**	**86.0**	48.0	**0.689**

4 EXPERIMENTS

We convert the first four layers of Llama-3.2-1B-Instruct (Grattafiori et al., 2024) to blockwise attention and add a neural memory module for each KV head. The remaining layers retain full attention. This setting tests whether memory can recover information no longer accessible through attention in the converted layers, without requiring every layer to rely on the new mechanism.

Each three-layer NMM learns historical key-value bindings using a self-supervised test-time objective. During post-training, we optimize the external modules for language modeling while keeping the pre-trained backbone frozen. We conduct a series of experiments to measure how much long-text performance the NMM recovers, which forms of recall it supports, and how our architectural design affects inference costs.

4.1 POST-TRAINING FOR LONG CONTEXT REASONING

We post-train Titan-Llama on long sequences using a standard next-token prediction loss, while the NMM continues to update its fast weights through the self-supervised key-value objective in Section 3.2. The language modeling loss trains the added components to make those memory reads useful to the frozen backbone. We initially considered distilling the backbone's historical attention output into the NMM, but found this difficult to scale and ineffective in practice. Under our final objective, $\hat{O}_{\text{hist}}$ need not reproduce that output exactly and instead learns a contribution that helps predict future tokens under segmented attention.

We show that training Titan-Llama on next-token prediction, the details of which can be found in Appendix A, improves not just next-token prediction but also recall and reasoning over long-term dependencies. We evaluate long-context question answering on QuALITY (Pang et al., 2022) and LongHealth (Adams et al., 2024). Results are shown in Figure 3 and compared against a segmented attention baseline to show the effect induced by adding the NMM.

Figure 2 shows how language-modeling loss and decode latency change with context length. Loss delta is the difference in per-token negative log-likelihood from full-attention Llama, so values below zero indicate better performance. With 1024-token segments, Titan-Llama stays below zero from 16K through 128K tokens, while the measured Mamba-2, Gated DeltaNet, and LaCT baselines incur larger loss penalties at longer contexts. Because the model is training on sequences of at most 16K tokens, the measurements beyond 16K also test how its language-modeling behavior extends beyond the training length.

Table 2: RULER accuracy by task family and context length. Titan-Llama recovers much of the accuracy lost to attention segmentation, although the gap to full attention widens on some tasks at longer contexts. Multi-key and FWE/CWE results exceed the full attention baseline.

Model	Single needle				Multi-key				Multi-query			
	4K	8K	16K	32K	4K	8K	16K	32K	4K	8K	16K	32K
Full attention	**100.0**	**99.3**	**100.0**	**99.3**	82.0	74.0	63.3	**57.3**	**99.5**	**94.0**	**94.0**	**83.5**
Segmented attn.	10.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Titan-Llama	97.3	96.0	88.0	78.7	**91.3**	**88.0**	**66.7**	39.3	96.5	90.5	70.5	34.5
Model	Multi-value				FWE/CWE				HotpotQA			
	4k	8k	16k	32k	4k	8k	16k	32k	4k	8k	16k	32k
Full attention	**99.5**	**97.5**	**89.0**	**70.0**	**59.1**	39.2	22.1	20.0	**48.0**	40.0	**46.0**	**36.0**
Segmented attn.	1.5	0.0	0.0	0.0	27.0	9.8	3.3	0.0	8.0	0.0	0.0	0.0
Titan-Llama	88.0	81.0	63.0	53.0	53.4	**44.9**	**32.7**	**24.8**	40.0	**46.0**	28.0	26.0

4.2 RULER Evaluation

Aggregate language-modeling loss does not tell us which kinds of historical information are best encoded by the NMM. The RULER benchmark measures different types of recall over long sequences, including retrieval of individual values, multiple associations, and word-frequency information (Hsieh et al., 2024). The RULER results in Table 2 compare full attention, segmented attention without memory, and Titan-Llama. The NMMs are able to effectively enrich representations enough to recover the majority of backbone performance, particularly for single-needle and multi-key retrieval at lengths up to 16K.

Unexpectedly, Titan-Llama also exceeds the full-attention baseline on multi-key retrieval at 4K–16K and on frequent/common word extraction at 8K–32K. Multi-value retrieval remains below full attention for every length, suggesting greater difficulty returning several distinct values from the same context. Performance declines at 32K, beyond the 16K training length, suggesting limited length generalization in this setting. This reveals that the NMM can perform semantic retrieval, accumulation, and recall of individual values, but is less proficient at recovering several values at once.

When compared to other efficient architectures in Table 1, Titan-Llama matches the full attention baseline on pass-key retrieval and exceeds full attention performance on UUID recall at 8K and 16K. On 32K pass-key retrieval, it also retains full accuracy where the measured efficient-model baselines decline. These comparisons show that compacting history can preserve strong retrieval on some tasks, even though the broader RULER results show that it does not preserve every form of recall equally well. Especially compared to RNN-based models with limited full-attention layers such as Mamba-2 and Gated DeltaNet, Titan-Llama's performance does not degrade as severely with increasing context lengths.

4.3 Inference Gains

While various studies have applied TTT to modulate existing architecture in transformer, full attention oftentimes serves as a lower bound on performance and a ceiling on inference efficiency. (Feng et al., 2026) Because our use of TTT adapts a cheaper replacement to long-context relationships, we are able to harness greater inference efficiency. The decode comparison in Figure 2 shows this in effect as Titan-Llama has lower median single-token latency than the measured full-attention and efficient-model baselines across the reported context lengths.

Figure 4 separately measures how these costs change as we convert more layers. At longer prompt lengths, Titan-Llama achieves lower prefill latency than the full-attention backbone, and the advantage grows with the number of converted layers, the details of which can be found in Appendix B. In fact, the prefill process for Titan-Llama is more than 20% faster than full-attention at long prefill lengths with only five converted layers. The steady-state decode memory usage for Titan-Llama is also 24% lower than full attention with the KV cache, though peak memory usage during prefill is more expensive for partial hybrids. This reflects our intuition that our approach should result in

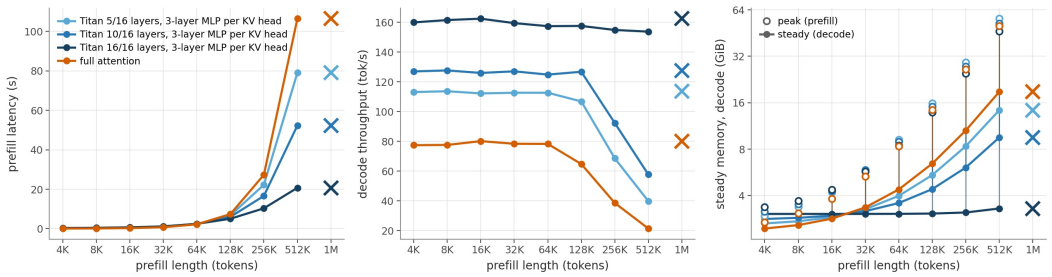

Figure 4: Increasing the proportion of Llama 3.2 1B layers retrofitted with NMMs increases token decode throughput, decreases prefill latency, and also decreases memory footprint during token decoding. Peak memory footprint during prefill for partial Titan-Llama hybrids exceeds full-attention Llama while retrofitting all layers leads to lower peak memory footprint. Benchmarking was performed on a single 80GB NVIDIA A100 GPU.

inference economics gains, with one practical limitation that the *peak* memory footprint of Titan-Llama exceeds that of full attention in shorter sequence lengths and in hybrid variants.

4.4 Scaling chunk size

The chunk size in Titan-Llama corresponds to the number of tokens in a local window as well as the effective token batch size for the inner objective loss. A similar hyperparameter affects not just inference dynamics but capabilities in other studies as well. Feng et al. (2026) Changing it affects both what information remains available through exact local attention and how often the NMM is updated. The RULER and long-context reasoning results seen previously utilized 512-token segments. Figure 3 varies the chunk size from 512 to 2048 tokens. Larger chunks improve accuracy on QuALITY and LongHealth, but their effects on RULER are mixed: they help some task–length combinations while reducing accuracy on others, including several needle retrieval settings.

We also study the effects of scaling context length on RULER performance and show in Appendix Table 3 that shorter context lengths lead to better performance on NIAH-style tasks while longer chunk sizes lend better semantic retrieval.

Larger chunks also increase measured prefill latency. Chunk size thus sets a task-dependent balance among local attention, the frequency of memory updates, and inference cost, rather than providing a uniform accuracy gain.

5 Discussion

The idea of using test-time training on an MLP to improve language model capabilities, whether on an existing part of LLM architecture or on external modules, has been well explored in a substantial body of work. (Behrouz et al., 2024; Feng et al., 2026; Tandon et al., 2025; Sun et al., 2025; Zhang et al., 2025) Compared to these studies, however, our main contribution seeks to take advantage of the theoretical understanding of the updated set of weights as a memory mechanism and use it to *substitute* the long-range attention calculations of a pre-trained backbone.

To that end, our results corroborate the current line of research which utilizes TTT to improve language modeling, while demonstrating practical inference-efficiency benefits. Similar to Zhang et al. (2025); Behrouz et al. (2024), we show that TTT using KV binding effectively learns to do language modeling. And like Feng et al. (2026), we show that TTT can extend the context window for certain tasks.

Our work also allows for open questions such as scaling NMM size or changing the NMM architecture. Further, understanding the mechanisms behind memory retrieval is a natural extension of this work, and exploring the use of inner-loop objectives other than KV binding could lead to improvements on this method.

AI USE STATEMENT

We used generative AI tools to proofread the manuscript and to assist in creating the schematic architecture figure. We did not use generative AI to generate research ideas, scientific claims, experimental results, or citations. The authors reviewed and revised all AI-assisted text and visual content, verified that the schematic accurately represents the proposed method, and checked the manuscript against the underlying experiments and cited sources. We take full responsibility for the final content of this work, including all text, claims, and artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We provide the code needed to train, evaluate, and benchmark Titan-Llama in the supplementary materials. The main paper describes the model architecture, learning objectives, evaluation settings, and reported baselines. Appendix A documents the datasets, training stages, hyperparameters, model configuration, and chunk sizes used in our experiments, while the inference implementation details are provided in the appendix. The released materials include instructions and configuration information necessary to reproduce the reported experiments and results

REFERENCES

Lisa Adams, Felix Busch, Tianyu Han, Jean-Baptiste Excoffier, Matthieu Ortala, Alexander Löser, Hugo JWL. Aerts, Jakob Nikolas Kather, Daniel Truhn, and Keno Bressem. Longhealth: A question answering benchmark with long clinical documents, 2024. URL https://arxiv.org/abs/2401.14490.

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. Etc: Encoding long and structured inputs in transformers, 2020. URL https://arxiv.org/abs/2004.08483.

Yushi Bai, Xin Lv, Jiajie Zhang, Yuze He, Ji Qi, Lei Hou, Jie Tang, Yuxiao Dong, and Juanzi Li. Longalign: A recipe for long context alignment of large language models, 2024. URL https://arxiv.org/abs/2401.18058.

Rachit Bansal, Aston Zhang, Rishabh Tiwari, Lovish Madaan, Sai Surya Duvvuri, Devvrit Khatri, David Brandfonbrener, David Alvarez-Melis, Prajjwal Bhargava, Mihir Sanjay Kale, and Samy Jelassi. Let's (not) just put things in context: Test-time training for long-context llms, 2025. URL https://arxiv.org/abs/2512.13898.

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

Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. Atlas: Learning to optimally memorize the context at test time, 2025. URL https://arxiv.org/abs/2505.23735.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

Aviv Bick, Kevin Y. Li, Eric P. Xing, J. Zico Kolter, and Albert Gu. Transformers to ssms: Distilling quadratic knowledge to subquadratic models, 2025. URL https://arxiv.org/abs/2408.10189.

Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Recurrent memory transformer, 2022. URL https://arxiv.org/abs/2207.06881.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024. URL https://arxiv.org/abs/2405.21060.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study, 2025. URL https://arxiv.org/abs/2506.06266.

Guhao Feng, Shengjie Luo, Kai Hua, Ge Zhang, Di He, Wenhao Huang, and Tianle Cai. In-place test-time training, 2026. URL https://arxiv.org/abs/2604.06169.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017. URL https://arxiv.org/abs/1703.03400.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao

Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752.

Dongchen Han, Yining Li, Tianyu Li, Zixuan Cao, Ziming Wang, Jun Song, Yu Cheng, Bo Zheng, and Gao Huang. Vit^3: Unlocking test-time training in vision, 2026. URL https://arxiv.org/abs/2512.01643.

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.

Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling, 2025. URL https://arxiv.org/abs/2507.07955.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL https://arxiv.org/abs/2006.16236.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020. URL https://arxiv.org/abs/2001.04451.

Yuto Kojima, Jiarui Xu, Xueyan Zou, and Xiaolong Wang. Lora-ttt: Low-rank test-time training for vision-language models, 2025. URL https://arxiv.org/abs/2502.02069.

Yuri Kuratov, Matvey Kairov, Aydar Bulatov, Ivan Rodkin, and Mikhail Burtsev. Gradmem: Learning to write context into memory with test-time gradient descent, 2026. URL https://arxiv.org/abs/2603.13875.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. Hierarchical transformers are more efficient language models, 2022. URL https://arxiv.org/abs/2110.13711.

Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, and Samuel R. Bowman. Quality: Question answering with long input texts, yes!, 2022. URL https://arxiv.org/abs/2112.08608.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting, 2024. URL https://arxiv.org/abs/2311.18677.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding, 2020. URL https://arxiv.org/abs/1911.02972.

Kevin Slagle. Spacebyte: Towards deleting tokenization from large language modeling, 2024. URL https://arxiv.org/abs/2404.14408.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts, 2020. URL https://arxiv.org/abs/1909.13231.

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, 2025. URL https://arxiv.org/abs/2407.04620.

Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, Carlos Guestrin, Jed McCaleb, Yejin Choi, and Yu Sun. End-to-end test-time training for long context, 2025. URL https://arxiv.org/abs/2512.23675.

Junxiong Wang, Daniele Paliotta, Avner May, Alexander M. Rush, and Tri Dao. The mamba in the llama: Distilling and accelerating hybrid models, 2025a. URL https://arxiv.org/abs/2408.15237.

Siyuan Wang, Gaokai Zhang, Li Lyna Zhang, Ning Shang, Fan Yang, Dongyao Chen, and Mao Yang. Loongrl: Reinforcement learning for advanced reasoning over long contexts, 2025b. URL https://arxiv.org/abs/2510.19363.

Qishuai Wen, Zhiyuan Huang, Xianghan Meng, Wei He, and Chun-Guang Li. Mixture-of-top-k attention: Efficient attention via scalable fast weights, 2026. URL https://arxiv.org/abs/2602.01219.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025. URL https://arxiv.org/abs/2412.06464.

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, 2025. URL https://arxiv.org/abs/2505.23884.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast kv compaction via attention matching, 2026. URL https://arxiv.org/abs/2602.16284.

A Training Details

A.1 Datasets

We train on one billion tokens of packed sequences from FineWeb, with a maximum sequence length of 16K tokens (Penedo et al., 2024). We then perform supervised fine-tuning on LongAlign (Bai et al., 2024) and LoongRL (Wang et al., 2025b). A final fine-tuning stage uses the QuALITY training set to adapt the model to multiple-choice answering (Pang et al., 2022).

A.2 Hyperparameters

The first four Llama layers each contain one neural memory module (NMM) per KV head. Each NMM is a three-layer network with a hidden dimension of 64 and GELU activations. The added components contain 17.23M trainable parameters in total.

We use a batch size of 64 during the initial training stage, corresponding to approximately one million tokens per batch at the maximum sequence length. Supervised fine-tuning uses a batch size of 128 with variable answer lengths. The inner-loss batch size equals the chunk size. We evaluate chunk sizes of 512, 1024, and 2048 tokens in Section 4.4. Compute constraints prevented a broad search over these architectural and training choices.

B Inference Implementation

Autoregressive decoding performs many small operations for each generated token. At batch size one, the cost of launching those operations can be substantial. For the measurements in Figure 4, we compile the Titan-Llama layers in CUDA Graph "reduce-overhead" mode and replay their single-token decode computations. Compilation and graph capture take place during warmup, outside the timed decode steps.

The captured computation uses a preallocated KV ring buffer for each attention segment. Cache positions and memory-store indices are updated in place on the GPU, and attention operands retain fixed shapes. We capture separate paths for ordinary tokens and for tokens that complete a memory-update chunk. The remaining full-attention layers are compiled without graph capture so that they retain their optimized attention implementation.

CUDA Graphs reduce launch overhead but still execute the underlying kernels. A fused implementation of the memory read, update, and output blend could reduce intermediate memory traffic and launch overhead further. Such an implementation would need to preserve the ordering of reads and updates at chunk boundaries.

Table 3: RULER accuracy by task family, context length, and chunk size. Bold marks the best result among the three chunk sizes for each task and length.

Chunk size	Single needle				Multi-key				Multi-query				Multi-value			
	4k	8k	16k	32k	4k	8k	16k	32k	4k	8k	16k	32k	4k	8k	16k	32k
512	**97.3**	**96.0**	88.0	**78.7**	**91.3**	**88.0**	66.7	**39.3**	**96.5**	**90.5**	70.5	34.5	**88.0**	**81.0**	63.0	**53.0**
1024	95.3	92.7	91.0	**78.7**	81.3	79.3	63.0	36.3	92.8	85.0	51.2	**42.2**	87.0	68.8	49.0	38.2
2048	90.3	92.7	**91.7**	51.7	66.3	74.0	**67.0**	32.3	91.3	88.8	**81.2**	19.2	87.5	79.2	**82.2**	25.0
Chunk size	FWE/CWE				HotpotQA				SQuAD				VT			
	4k	8k	16k	32k	4k	8k	16k	32k	4k	8k	16k	32k	4k	8k	16k	32k
512	53.4	**44.9**	32.7	**24.8**	40.0	**46.0**	28.0	26.0	55.5	40.2	**36.2**	21.5	**66.8**	**68.0**	**33.6**	9.2
1024	54.0	38.7	26.5	21.0	**43.0**	37.0	34.0	**29.0**	62.4	39.4	32.1	**31.0**	49.4	37.6	17.0	**32.2**
2048	**55.1**	36.5	**32.9**	22.4	40.0	33.0	**36.0**	**29.0**	**67.1**	**44.1**	32.8	27.7	58.2	46.6	12.2	18.0

C Additional Experiments

C.1 Chunk Size and RULER Performance

Chunk size controls both the span of local attention and the number of tokens used for each memory update. Table 3 reports RULER results for 512-, 1024-, and 2048-token chunks. The effect depends on the task and context length. The 512- token setting is strongest on most multi-key and multi-value retrieval settings, while larger chunks improve some individual results, including 16K SQuAD and multi-value accuracy. These results do not support a single best chunk size across the RULER tasks.