Fast-weight Product Key Memory

Tianyu Zhao [1] **Llion Jones** [1]

Abstract

Foundation models need mechanisms for rapid continual adaptation without repeatedly updating all slow weights. We introduce Fast-weight Product Key Memory (FwPKM), a sparse fast-weight memory layer for language modeling. FwPKM transforms Product Key Memory from a static slow-weight module into an online-updated episodic memory: it keeps the sparse PKM retrieval process, but updates activated key–value parameters at both training and inference time using chunk-level gradient descent on a local memory-rewrite objective. This performs Test-Time Training (TTT)-style updates over a sparse memory, enabling many context-specific associations to be memorized and retrieved with fixed per-token compute. Experiments show that FwPKM complements standard slow-weight modules, improves long-context perplexity, and generalizes to 128K-token Needle-in-a-Haystack contexts despite being trained on only 4K-token sequences. In online domain adaptation, FwPKM adapts quickly but exposes retention challenges, motivating future work on memory consolidation.

1. Introduction

Long-context language modeling often requires remembering specific, local facts (*e.g.* a name, a variable binding, a constraint) introduced tens of thousands of tokens earlier, and retrieving them when needed. The core sequence modeling layers, or token mixers, can be viewed as *associative memory* systems (Vaswani et al., 2017; Yang et al., 2024b; 2025; Peng et al., 2025; Gu & Dao, 2024; Dao & Gu, 2024; Sun et al., 2025; Behrouz et al., 2025c), where information from past tokens is encoded into storage (*i.e. memorization*) and later integrated into prediction by querying that storage (*i.e. retrieval*).

[1]Sakana AI, Japan. Correspondence to: Tianyu Zhao <tianyu@sakana.ai>.

Presented at the ICML 2026 Workshop "Continual Adaptation at Scale: Towards Sustainable AI". Copyright 2026 by the author(s).

The current landscape is defined by a trade-off between memory capacity and computational efficiency. Softmax self-attention (Vaswani et al., 2017; Zhong et al., 2025) explicitly stores a key-value pair for every past token, yielding high-fidelity retrieval but quadratic cost. In contrast, modern Recurrent Neural Networks (RNNs) such as linear attention (Katharopoulos et al., 2020), Mamba2 (Dao & Gu, 2024), and DeltaNet (Schlag et al., 2021a; Yang et al., 2024b) compress history into a fixed-size state, enabling efficient sub-quadratic retrieval but limiting how much information the model can retain.

Recent work on Test-Time Training (TTT, Sun et al. 2025; Zhang et al. 2025b) and Titans (Behrouz et al., 2025c) addresses this limitation by replacing the fixed state matrix with a "fast-weight" (Hinton & Plaut, 1987; Schmidhuber, 1992) neural network that adapts to input sequences on-the-fly. However, these approaches face a scaling bottleneck: dense network structure. To store the massive amount of information required for long-context tasks, the fast-weight network must be large; yet updating and querying a large dense network frequently is computationally expensive.

In this paper, we show that sparse associative memory offers a way around this constraint. Sparse memory architectures such as Product Key Memory (PKM, Lample et al. 2019) can maintain a very large number of memory parameters while activating only a small subset per token. We introduce Fast-weight Product Key Memory (FwPKM), which transforms PKM from a static slow-weight module into a dynamic fast-weight episodic memory. By updating sparse key/value matrices via gradient descent at training and inference time, FwPKM can memorize many context-specific associations with fixed per-token compute.

Empirically, FwPKM improves long-context perplexity and complements standard slow-weight modules. In Needle-in-a-Haystack evaluations, FwPKM generalizes to 128K-token contexts despite being trained on only 4K-token sequences. We further study online domain adaptation, finding that FwPKM quickly adapts to new domains but faces retention challenges, motivating future work on memory consolidation.

2. Fast-weight Product Key Memory

2.1. Product Key Memory Retrieval

We briefly review Product Key Memory (PKM, Lample et al. 2019), which we later extend into an online-updated fast-weight module. A key-value memory stores keys $K \in \mathbb{R}^{N \times d_k}$ and values $V \in \mathbb{R}^{N \times d_v}$. Given a query $\mathbf{q}$, dense Top-k memory scores all N slots and retrieves a weighted sum over the selected values. PKM reduces this scoring cost by factorizing the key space into two smaller codebooks. It splits the query into two sub-queries $\mathbf{q} = [\mathbf{q}^{(1)}; \mathbf{q}^{(2)}]$, where $\mathbf{q}^{(1)}, \mathbf{q}^{(2)} \in \mathbb{R}^{d_k/2}$, and maintains two sub-key matrices $K^{(1)}, K^{(2)} \in \mathbb{R}^{\sqrt{N} \times d_k/2}$. Each codebook is scored independently:

$$s_i^{(m)} = \mathbf{q}^{(m)\top} K_i^{(m)}, \quad m \in \{1, 2\}. \tag{1}$$

PKM selects Top-k indices from each codebook, forms the restricted Cartesian product $\mathcal{I}^{(1)} \times \mathcal{I}^{(2)}$, and scores each pair additively:

$$s_{i,j} = s_i^{(1)} + s_j^{(2)}, \quad (i, j) \in \mathcal{I}^{(1)} \times \mathcal{I}^{(2)}. \tag{2}$$

Finally, it selects the Top-k pairs, applies softmax over their scores, and retrieves from the corresponding rows of V, where pair (i, j) maps to row $((i-1)\sqrt{N} + j)$. This decomposition avoids scoring all N slots and allows PKM to index millions of values with only $O(\sqrt{N})$ key scores.

2.2. FwPKM: Online-Updating Sparse Memory

Standard PKM is a **slow-weight** sparse channel mixer: its parameters are learned by the global language modeling objective and remain frozen at inference. As a result, PKM mainly stores *semantic* knowledge but cannot rapidly incorporate *episodic* information from the current input stream. We propose **Fast-weight Product Key Memory** (FwPKM), which keeps PKM's **retrieval process unchanged**, but redesigns the **memorization process** by updating the memory matrices online. Concretely, FwPKM stores key-value associations from the input sequence by minimizing local reconstruction objectives over chunks, in the spirit of test-time training.

Given hidden states $\mathbf{h}_{1:T}$, we process them in chunks of size C. For each token t, a slow-weight network parameterized by ϕ constructs a query, a value, and a scalar gate:

$$\mathbf{q}_t, \mathbf{v}_t = \mathtt{Linear}_\phi^{q,v}(\mathtt{RMSNorm}_\phi^{q,v}(\mathbf{h}_t)), \tag{3}$$

$$g_t = \mathtt{Sigmoid}\Big(\mathtt{Linear}_\phi^{g}(\mathtt{RMSNorm}_\phi^{g}(\mathbf{h}_t))\Big). \tag{4}$$

FwPKM then uses PKM retrieval with the current fast weights $\theta = \{K^{(1)}, K^{(2)}, V\}$:

$$\hat{\mathbf{v}}_t = \mathtt{PKM}(\mathbf{q}_t; \theta) = \sum_{i \in \mathcal{I}_t} s'_{t,i} V_i. \tag{5}$$

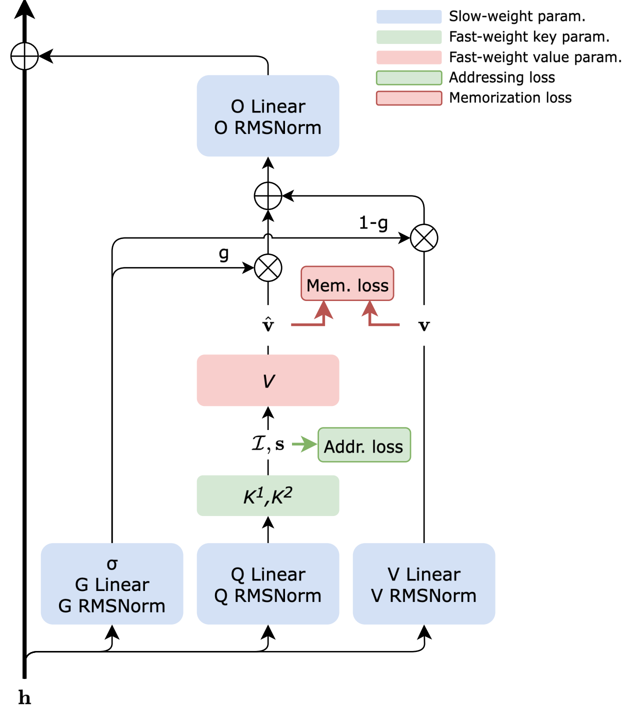

Figure 1. **Architecture of FwPKM.** Slow weights are updated only by the global LM loss, while fast-weight keys and values are updated at training and inference time by local losses.

The retrieved value is interpolated with a residual value:

$$\mathbf{o}_t = g_t \cdot \hat{\mathbf{v}}_t + (1 - g_t) \cdot \mathbf{v}_t, \tag{6}$$

$$\mathbf{o}'_t = \mathtt{Linear}_\phi^{o}(\mathtt{RMSNorm}_\phi^{o}(\mathbf{o}_t)). \tag{7}$$

We apply fast-weight updates after processing each chunk, so predictions within a chunk depend only on fast weights from previous chunks, preserving causality.

2.3. Fast-weight Memorization

Within each chunk, FwPKM updates fast weights to make retrieval reconstruct target values derived from the same stream. For a chunk of size C, we minimize a gated reconstruction loss:

$$\mathcal{L}_{\text{mem}} = \sum_{t=1}^{C-1} \frac{1}{2} g_t \, \|\mathbf{v}_t^{\text{tar}} - \hat{\mathbf{v}}_t\|_2^2. \tag{8}$$

In our experiments, we use lookahead targets $\mathbf{v}_t^{\text{tar}} = \mathbf{v}_{t+1}$, which aligns the local memorization objective with next-token prediction. We update only the accessed value rows. When multiple tokens write to the same row, we normalize the aggregated gradient by the number of accesses N_i^{read}:

$$\nabla_{V_i}^{\text{agg}} = \frac{1}{N_i^{\text{read}}} \nabla_{V_i} \mathcal{L}_{\text{mem}}, \qquad V_i \leftarrow V_i - \nabla_{V_i}^{\text{agg}}. \tag{9}$$

This acts as a simple consensus mechanism when competing tokens write to the same memory row.

2.4. Addressing Optimization

Sparse memories can suffer from *memory slot collapsing*, where only a small fraction of slots are used. We therefore optimize an auxiliary addressing objective that encourages balanced slot usage on average across a chunk, without forcing each individual query to be uniform. Let $\mathbf{s}_t'^1, \mathbf{s}_t'^2 \in \mathbb{R}^{\sqrt{N}}$ be the normalized Top-k score vectors over the two sub-key sets, with unselected indices set to 0. We define marginal slot-usage distributions

$$\bar{\mathbf{p}}^m = \frac{1}{C}\sum_{t=1}^{C} \mathbf{s}_t'^m, \quad m \in \{1, 2\}, \tag{10}$$

and minimize the negative entropy:

$$\begin{aligned} \mathcal{L}_{\text{addr}} &= -H(\bar{\mathbf{p}}^1) - H(\bar{\mathbf{p}}^2) \\ &= -\sum_{i=1}^{\sqrt{N}} \bar{p}_i^1 \log \bar{p}_i^1 - \sum_{i=1}^{\sqrt{N}} \bar{p}_i^2 \log \bar{p}_i^2. \end{aligned} \tag{11}$$

The keys are updated by

$$K^m \leftarrow K^m - \nabla_{K^m} \mathcal{L}_{\text{addr}}, \quad m \in \{1, 2\}. \tag{12}$$

For stability, we also use inverse-distance weighting scores and target normalization; details and ablations are deferred to Appendix B and Appendix G.

3. Experiments

Setup. In the main content, we focus on 12-layer QwenNext-style language models (Qwen Team, 2025b;a), which interleaves Gated DeltaNet (Yang et al., 2025) and Sliding Window Attention with window size 512 at a 3:1 ratio, and compare with LaCT (Zhang et al., 2025b) and MoM (Du et al., 2026). The models are trained on 5B tokens from LongContext64 (Buckman) and 5B tokens from Fineweb-Edu (Penedo et al., 2024), with sequence length 4K. PKM and FwPKM modules use 512^2 memory slots and are inserted at layers 2, 6, and 10 unless otherwise stated. Additional architecture and optimization details are in Appendix D.

Perplexity and LongBench. We evaluate perplexity on Fineweb-Edu, LC64, and LAMBADA (Paperno et al., 2016). Each evaluation set contains 8M tokens. For FwPKM, 4K-token segments are evaluated in their original order with batch size 1, without resetting fast weights between adjacent segments, allowing fast weights to capture cross-segment dependencies. We also evaluate answer-token perplexity on five LongBench (Bai et al., 2024) tasks: *2WikiMultihopQA*, *HotpotQA*, *MultiFieldQA-en*, *MuSiQue*, and *NarrativeQA*. Table 1 reports the average over these five tasks; the per-task numbers are in Appendix E.

Table 1. Main results for the `GDN+SWA` setting. FW: Fineweb-Edu; LC64: LongContext64; LMB: LAMBADA; LB Avg.: average answer-token PPL over five LongBench tasks. Lower is better.

Model	FW	LC64	LMB	LB Avg.
`GDN+SWA`	18.00	9.20	29.36	41.10
`+ PKM@2,6,10`	16.98	8.62	27.55	38.02
`+ FwPKM@2,6,10`	17.21	8.53	**26.11**	**26.49**
`+ PKM@6 + FwPKM@2,10`	**16.76**	**8.27**	26.43	30.00
`LaCT`	17.78	9.07	27.66	67.92
`MoM, 4 experts`	17.60	8.94	26.17	34.28
`MoM, 8 experts`	17.49	8.87	26.48	29.80
`MoM, 12 experts`	17.67	8.99	26.81	34.00

FwPKM substantially improves long-context perplexity. Compared with the `GDN+SWA` baseline, FwPKM reduces PPL on LC64 and LAMBADA, and gives the best LongBench average. Standard PKM gives the largest gain on Fineweb-Edu, while the hybrid `PKM@6 + FwPKM@2,10` model performs best on FW and LC64. This supports the view that slow-weight PKM and fast-weight FwPKM play complementary semantic- and episodic-memory roles.

Needle-in-a-Haystack. We use Needle-in-a-Haystack (NIAH, Kamradt 2023; Mohtashami & Jaggi 2023) to directly test episodic retrieval. Each example inserts key-value needles into a LAMBADA haystack, and the model must recover the value associated with a queried key. We evaluate context lengths from 4K to 128K. For FwPKM, fast weights are updated after reading the haystack, and we also test iterative rereading, where the same haystack is processed multiple times before answering.

The NIAH results in Figure 2 show two key behaviors. First, FwPKM generalizes beyond the 4K training length and remains effective at 128K-token contexts. Second, iterative rereading improves retrieval accuracy, indicating that repeated test-time updates can consolidate episodic memories.

Online domain adaptation. Finally, we evaluate whether FwPKM can adapt its fast weights to new domains without updating slow weights. We sequentially adapt on six Pile (Gao et al., 2020) domains – namely *DM Mathematics*, *FreeLaw*, *PhilPapers*, *PubMed Central*, *Ubuntu IRC*, and *USPTO Backgrounds* – and evaluate negative log-likelihood before and after each stage. As shown in Figure 3, FwPKM quickly reduces NLL on the currently adapted domain, but the gains often decay after later domains are processed. This suggests that FwPKM already provides rapid online adaptation, while long-term retention requires future memory-consolidation mechanisms. Note that baseline models do not show any decrease of NLL due to the lack or the limited capacity of fast weights. Full adaptation curves are shown in Appendix E.

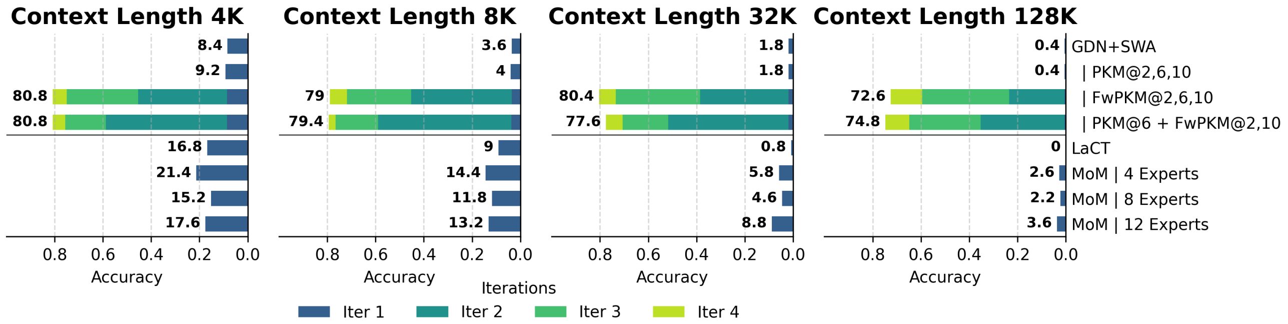

Figure 2. Needle-in-a-Haystack accuracy with iterative rereading of the haystack context. Stacked colors indicate accuracy after 1–4 rereading iterations. Higher is better.

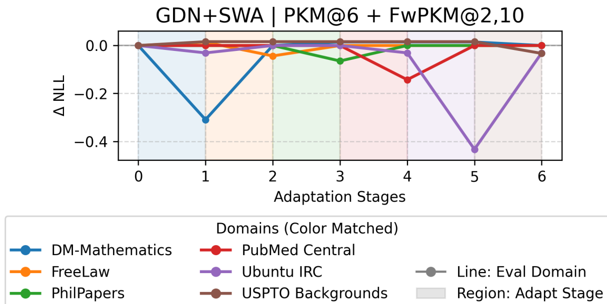

Figure 3. Change of negative log-likelihood on six Pile domains before and after 6 adaptation stages of test-time learning on each.

4. Related Work

Fast weights and test-time adaptation. Fast weights provide a classical view of sequence models as systems that rapidly update temporary parameters (Schmidhuber, 1992; Ba et al., 2016; Schlag et al., 2021a). Recent Test-Time Training (TTT) methods (Sun et al., 2025; Zhang et al., 2025b) and Titans (Behrouz et al., 2025c) revive this idea by updating model components during inference to memorize the current context. Broader frameworks such as MIRAS (Behrouz et al., 2025a) and Test-Time Regression (Wang et al., 2025) further cast sequence modeling as associative memory with test-time optimization. FwPKM follows this line, but replaces dense fast-weight networks with a sparse memory architecture whose active computation is decoupled from total storage capacity.

Sparse and hybrid memory architectures. Softmax attention (Vaswani et al., 2017; Zhong et al., 2025) provides high-fidelity retrieval but scales quadratically with context length, motivating efficient recurrent, linear-attention, and SSM-style models (Katharopoulos et al., 2020; Gu & Dao, 2024; Dao & Gu, 2024; Schlag et al., 2021b; Yang et al., 2024b; 2025; Peng et al., 2025). Hybrid architectures combine these complementary mechanisms (Irie et al., 2025; Ren et al., 2025; Team et al., 2025; Qwen Team, 2025b;a). Separately, explicit sparse memory modules such as Product Key Memory (Lample et al., 2019; Berges et al., 2025), PEER (He, 2024), and Ultra Sparse Memory (Huang et al., 2025b;a) scale storage by activating only a small subset of memory slots. Our work builds directly on PKM, but changes it from a static slow-weight module into an online-updated fast-weight episodic memory.

Continual and episodic learning. Updating memory parameters online connects FwPKM to continual learning and adaptation. Prior work has explored sparse slow-weight adaptation for mitigating forgetting (Lin et al., 2025), while Nested Learning (Behrouz et al., 2025b) and TNT (Li et al., 2025) study multi-timescale fast-weight systems for retention and consolidation. FwPKM is complementary: it maintains a large sparse episodic memory bank updated at chunk level, enabling rapid adaptation while exposing retention challenges that motivate future consolidation mechanisms.

5. Conclusion

We introduced **Fast-weight Product Key Memory** (FwPKM), a sparse fast-weight memory layer that turns Product Key Memory from a static slow-weight retrieval module into an online-updated episodic memory. By combining PKM's large sparse storage with TTT-style local updates, FwPKM addresses the trade-off between memory capacity and computational efficiency: it can store many context-specific associations while activating only a small subset per token. Experiments show that FwPKM improves long-context modeling, complements slow-weight PKM, generalizes to 128K-token Needle-in-a-Haystack contexts after training on 4K sequences, and supports rapid online domain adaptation. At the same time, our results reveal open challenges in update efficiency and long-term retention, suggesting future work on hardware-aware sparse update kernels and memory consolidation for scalable continual adaptation.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. In *NeurIPS*, 2016.

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., Dong, Y., Tang, J., and Li, J. LongBench: A bilingual, multitask benchmark for long context understanding. In *ACL*, 2024.

Behrouz, A., Razaviyayn, M., Zhong, P., and Mirrokni, V. It's all connected: A journey through test-time memorization, attentional bias, retention, and online optimization, 2025a. URL https://arxiv.org/abs/2504.13173.

Behrouz, A., Razaviyayn, M., Zhong, P., and Mirrokni, V. Nested learning: The illusion of deep learning architectures. In *NeurIPS*, 2025b. URL https://openreview.net/forum?id=nbMeRvNb7A.

Behrouz, A., Zhong, P., and Mirrokni, V. Titans: Learning to memorize at test time. In *NeurIPS*, 2025c. URL https://openreview.net/forum?id=8GjSf9Rh7Z.

Berges, V., Oguz, B., Haziza, D., Yih, W., Zettlemoyer, L., and Ghosh, G. Memory layers at scale. In *ICML*, 2025.

Buckman, J. LongCrawl64: A long-context natural-language dataset.

Csordás, R., Irie, K., and Schmidhuber, J. Approximating two-layer feedforward networks for efficient Transformers. In *Findings of EMNLP*. Association for Computational Linguistics, 2023.

Dao, T. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *ICLR*, 2024.

Dao, T. and Gu, A. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *ICML*, 2024.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*, 2022.

Du, J., Sun, W., Lan, D., Hu, J., and Cheng, Y. Mom: Linear sequence modeling with mixture-of-memories. In *ICLR*, 2026.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S., and Leahy, C. The Pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. In *COLM*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

He, X. O. Mixture of a million experts, 2024. URL https://arxiv.org/abs/2407.04153.

Hinton, G. E. and Plaut, D. C. Using fast weights to deblur old memories. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 177–186, 1987.

Huang, Z., Bao, Y., Min, Q., Chen, S., Guo, R., Huang, H., Zhu, D., Zeng, Y., Wu, B., Zhou, X., and Qiao, S. UltraMemV2: Memory networks scaling to 120b parameters with superior long-context learning, 2025a. URL https://arxiv.org/abs/2508.18756.

Huang, Z., Min, Q., Huang, H., Zeng, Y., Zhu, D., Guo, R., and zhou Xun. Ultra-Sparse Memory Network. In *ICLR*, 2025b.

Irie, K., Yau, M., and Gershman, S. J. Blending complementary memory systems in hybrid quadratic-linear Transformers. In *NeurIPS*, 2025.

Jordan, K., Jin, Y., Boza, V., You, J., Cesista, F., Newhouse, L., and Bernstein, J. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Kamradt, G. Needle In A Haystack – pressure testing LLMs, 2023. URL https://github.com/gkamradt/LLMTest_NeedleInAHaystack.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are RNNs: Fast autoregressive Transformers with linear attention. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020.

Lample, G., Sablayrolles, A., Ranzato, M., Denoyer, L., and Jégou, H. Large memory layers with product keys. In *NeurIPS*, 2019.

Li, Z., Behrouz, A., Deng, Y., Zhong, P., Kacham, P., Karami, M., Razaviyayn, M., and Mirrokni, V. TNT: Improving chunkwise training for test-time memorization, 2025. URL https://arxiv.org/abs/2511.07343.

Lin, J., Zettlemoyer, L., Ghosh, G., Yih, W.-T., Markosyan, A., Berges, V.-P., and Oğuz, B. Continual learning via sparse memory finetuning, 2025. URL https://arxiv.org/abs/2510.15103.

McCarter, C. Inverse distance weighting attention. *arXiv preprint arXiv:2310.18805*, 2023. URL https://arxiv.org/abs/2310.18805.

Mohtashami, A. and Jaggi, M. Landmark Attention: Random-access infinite context length for Transformers. In *NeurIPS*, 2023.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *ACL*. The Association for Computer Linguistics, 2016.

Penedo, G., Kydlícek, H., Allal, L. B., Lozhkov, A., Mitchell, M., Raffel, C. A., von Werra, L., and Wolf, T. The FineWeb datasets: Decanting the web for the finest text data at scale. In *NeurIPS*, 2024.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Derczynski, L., Du, X., Grella, M., GV, K. K., He, X., Hou, H., Kazienko, P., Kocon, J., Kong, J., Koptyra, B., Lau, H., Lin, J., Mantri, K. S. I., Mom, F., Saito, A., Song, G., Tang, X., Wind, J. S., Wozniak, S., Zhang, Z., Zhou, Q., Zhu, J., and Zhu, R. RWKV: Reinventing RNNs for the Transformer era. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of EMNLP*. Association for Computational Linguistics, 2023.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. YaRN: Efficient context window extension of large language models. In *ICLR*, 2024.

Peng, B., Zhang, R., Goldstein, D., Alcaide, E., Du, X., Hou, H., Lin, J., Liu, J., Lu, J., Merrill, W., Song, G., Tan, K., Utpala, S., Wilce, N., Wind, J. S., Wu, T., Wuttke, D., and Zhou-Zheng, C. RWKV-7 "Goose" with expressive dynamic state evolution. In *COLM*, 2025. URL https://openreview.net/forum?id=ayB1PACN5j.

Qwen Team. Qwen3-Next-80B-A3B-Thinking. https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Thinking, 2025a.

Qwen Team. Qwen3 technical report, 2025b. URL https://arxiv.org/abs/2505.09388.

Ren, L., Liu, Y., Lu, Y., Shen, Y., Liang, C., and Chen, W. Samba: Simple hybrid state space models for efficient unlimited context language modeling. In *ICLR*, 2025.

Schlag, I., Irie, K., and Schmidhuber, J. Linear Transformers are secretly fast weight programmers. In Meila, M. and Zhang, T. (eds.), *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9355–9366. PMLR, 2021a.

Schlag, I., Munkhdalai, T., and Schmidhuber, J. Learning associative inference using fast weight memory. In *ICLR*, 2021b.

Schmidhuber, J. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Comput.*, 4(1):131–139, 1992.

Sun, Y., Li, X., Dalal, K., Xu, J., Vikram, A., Zhang, G., Dubois, Y., Chen, X., Wang, X., Koyejo, S., Hashimoto, T., and Guestrin, C. Learning to (learn at test time): RNNs with expressive hidden states. In *ICML*, 2025.

Team, K., Zhang, Y., Lin, Z., Yao, X., Hu, J., Meng, F., Liu, C., Men, X., Yang, S., Li, Z., Li, W., Lu, E., Liu, W., Chen, Y., Xu, W., Yu, L., Wang, Y., Fan, Y., Zhong, L., Yuan, E., Zhang, D., Zhang, Y., Liu, T. Y., Wang, H., Fang, S., He, W., Liu, S., Li, Y., Su, J., Qiu, J., Pang, B., Yan, J., Jiang, Z., Huang, W., Yin, B., You, J., Wei, C., Wang, Z., Hong, C., Chen, Y., Chen, G., Wang, Y., Zheng, H., Wang, F., Liu, Y., Dong, M., Zhang, Z., Pan, S., Wu, W., Wu, Y., Guan, L., Tao, J., Fu, G., Xu, X., Wang, Y., Lai, G., Wu, Y., Zhou, X., Yang, Z., and Du, Y. Kimi Linear: An expressive, efficient attention architecture, 2025. URL https://arxiv.org/abs/2510.26692.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NeurIPS*, 2017.

Wang, K. A., Shi, J., and Fox, E. B. Test-time regression: A unifying framework for designing sequence models with associative memory, 2025. URL https://arxiv.org/abs/2501.12352.

Weston, J., Chopra, S., and Bordes, A. Memory networks. In *ICLR*, 2015.

Yang, S. and Zhang, Y. FLA: A Triton-based library for hardware-efficient implementations of linear attention mechanism, January 2024. URL https://github.com/fla-org/flash-linear-attention.

Yang, S., Wang, B., Shen, Y., Panda, R., and Kim, Y. Gated linear attention Transformers with hardware-efficient training. In *ICML*, 2024a.

Yang, S., Wang, B., Zhang, Y., Shen, Y., and Kim, Y. Parallelizing linear Transformers with the delta rule over sequence length. In *NeurIPS*, 2024b.

Yang, S., Kautz, J., and Hatamizadeh, A. Gated delta networks: Improving Mamba2 with delta rule. In *ICLR*, 2025.

Zhang, J. and Bottou, L. Memory mosaics at scale. In *NeurIPS*, 2025. URL https://openreview.net/forum?id=IfD2MKTmWv.

Zhang, J., Nolte, N., Sadhukhan, R., Chen, B., and Bottou, L. Memory mosaics. In *ICLR*, 2025a.

Zhang, T., Bi, S., Hong, Y., Zhang, K., Luan, F., Yang, S., Sunkavalli, K., Freeman, W. T., and Tan, H. Test-time training done right, 2025b. URL https://arxiv.org/abs/2505.23884.

Zhong, S., Xu, M., Ao, T., and Shi, G. Understanding Transformer from the perspective of associative memory, 2025. URL https://arxiv.org/abs/2505.19488.

A. Extended Related Work

Softmax attention and efficient sequence models. Softmax attention underlies Transformers (Vaswani et al., 2017) and can be viewed as associative memory (Zhong et al., 2025), but its quadratic cost limits very long contexts. Efficient alternatives include linear-attention-style models (Katharopoulos et al., 2020) and recurrent/SSM variants such as Mamba (Gu & Dao, 2024; Dao & Gu, 2024), DeltaNet and its gated/parallel forms (Schlag et al., 2021b; Yang et al., 2024b; 2025), and RWKV7 (Peng et al., 2025). Other approaches retain attention while improving memory efficiency via smoothing or hierarchical designs, e.g., Memory Mosaics (Zhang et al., 2025a; Zhang & Bottou, 2025).

Fast weights and test-time optimization. Fast weights provide a unifying view of sequence models (Schmidhuber, 1992; Ba et al., 2016; Schlag et al., 2021a). Recent Test-Time Training (TTT) (Sun et al., 2025; Zhang et al., 2025b) and Titans (Behrouz et al., 2025c) revive this idea by updating parameters at inference with gradient descent to memorize the current context. Broader perspectives such as MIRAS (Behrouz et al., 2025a) and Test-Time Regression (Wang et al., 2025) cast sequence models as associative memory with test-time optimization, emphasizing design choices in memory architecture, update rules, retention, and optimizers. FwPKM contributes a sparse memory architecture together with a chunk-level memorization rule compatible with sparse access.

Hybrid architectures. Hybrid designs interleave quadratic attention with efficient recurrent/linear layers to balance retrieval fidelity and cost, including models trained from scratch (Irie et al., 2025) and large-scale hybrids such as Samba (Ren et al., 2025), KimiLinear (Team et al., 2025), and QwenNext (Qwen Team, 2025b;a). Our experiments similarly combine linear attention (GDN), softmax attention, slow-weight sparse memory (PKM), and fast-weight sparse memory (FwPKM) to form complementary memory components.

Explicit sparse memory modules. Beyond implicit knowledge in weights, explicit memory has been explored to expand storage capacity (Weston et al., 2015), and even simple MLPs can exhibit memory-like behavior (Csordás et al., 2023). To scale capacity without prohibitive cost, sparse access is crucial: PKM and related modules (Lample et al., 2019; Berges et al., 2025; He, 2024; Huang et al., 2025b;a) retrieve from massive memory banks efficiently. Our work builds on the PKM structure but transitions it from a static slow-weight module into a dynamic fast-weight memory updated online.

Continual and episodic learning. Updating memory parameters naturally connects to continual learning and adaptation. Lin et al. (2025) showed that parameter-efficient fine-tuning of PKM can mitigate catastrophic forgetting, updating *semantic* memory via sparse slow weights. FwPKM opens a complementary axis: online updates to *episodic* memory through fast weights. Nested Learning (Behrouz et al., 2025b) and TNT (Li et al., 2025) explore multi-timescale fast-weight systems by stacking fast-weight layers (e.g., Titans) and updating at varying frequencies, enabling consolidation from faster to slower memories. FwPKM instead maintains a large episodic memory bank updated at low frequency to amortize optimization cost, suggesting future hybrid systems that combine varying-size FwPKM*s* with other components (e.g., Titans) under different optimization and retention strategies.

B. FwPKM Implementation Details

B.1. Sparse Top-k Key-Value Memory

A standard key-value memory consists of a key matrix $K \in \mathbb{R}^{N \times d_k}$ and a value matrix $V \in \mathbb{R}^{N \times d_v}$, where N is the number of memory slots and $d_{\{k,v\}}$ are the hidden dimensions. Given a query vector $\mathbf{q} \in \mathbb{R}^{d_k}$, the model scores each slot (*e.g.*, via dot product),

$$s_i = \mathbf{q}^\top K_i, \tag{13}$$

selects the indices of the Top-k scores $\mathcal{I}$,

$$\mathcal{I} = \texttt{Top-}k(\{s_i\}_{i=1}^{N}), \tag{14}$$

computes normalized scores $\{s'_i\}$ via softmax,

$$\{s'_i\} = \texttt{softmax}(\{s_j\}_{j \in \mathcal{I}}), \tag{15}$$

and retrieves a weighted sum of values,

$$\hat{\mathbf{v}} = \textstyle\sum_{i \in \mathcal{I}} s'_i V_i. \tag{16}$$

Top-k sparsity reduces the number of *accessed* value rows, but still requires *scoring* all N keys to identify the top candidates. This linear $O(N)$ complexity prohibits scaling to massive memory sizes (*e.g.* $N \approx 10^6$). Product Key Memory reduces this scoring cost through the factorized retrieval process described in Section 2.1.

B.2. MSE as an Explicit Rewrite Signal

The MSE gradient provides an explicit "rewrite" signal. For a single prediction-target pair $(\hat{\mathbf{v}}, \mathbf{v})$, the gradient with respect to the prediction is

$$\nabla_{\hat{\mathbf{v}}} \frac{1}{2}\|\mathbf{v} - \hat{\mathbf{v}}\|_2^2 = -(\mathbf{v} - \hat{\mathbf{v}}). \tag{17}$$

Thus, if we could update the prediction directly, a unit step would rewrite it to the target:

$$\hat{\mathbf{v}} \leftarrow \hat{\mathbf{v}} - \nabla_{\hat{\mathbf{v}}} = \mathbf{v}. \tag{18}$$

In FwPKM, we do not update $\hat{\mathbf{v}}$ directly; instead we update the underlying value rows V_i that produced $\hat{\mathbf{v}}$. Nevertheless, the same residual term $(\mathbf{v} - \hat{\mathbf{v}})$ acts as the update signal, making the fast-weight update behave like explicit memory rewriting.

B.3. Lookahead Value Targets

By default, FwPKM pairs each query $\mathbf{q}_t$ with the value computed from the same token, $\mathbf{v}_t$, in the memorization loss. However, next-token prediction benefits more directly from information about *future* tokens. We therefore use a **lookahead** construction that pairs $\mathbf{q}_t$ with the next token's value $\mathbf{v}_{t+1}$ when applying chunk-level updates.

Concretely, we keep the retrieval definition $\hat{\mathbf{v}}_t = \texttt{PKM}(\mathbf{q}_t; \theta)$ unchanged, and only shift the target value in the memorization objective. We use gradients from this new objective to update the fast-weight value matrix V:

$$\mathcal{L}_{\text{mem}}^{\text{LA}} = \sum_{t=1}^{C-1} \frac{1}{2}\, g_t \,\|\mathbf{v}_{t+1} - \hat{\mathbf{v}}_t\|_2^2. \tag{19}$$

Intuitively, this encourages FwPKM to store information that is immediately useful for predicting the next token, similar in spirit to using short convolutions in linear-attention variants (Peng et al., 2023; Yang et al., 2024a; Gu & Dao, 2024).

Importantly, we apply the fast-weight update only after processing the entire chunk, so lookahead targets do not affect predictions within the same chunk. We carefully handle values at chunk boundaries. For efficiency, we drop the last predicted value in each chunk *at training time*, which results in dropping 8 tokens for a 4096-token training sequence. At inference time, FwPKM maintains a value cache that accumulates new $(\mathbf{v}_{t+1}, \hat{\mathbf{v}}_t)$ pairs. When a pre-defined update chunk size is reached (*e.g.* 512 in PPL evaluation, haystack length in NIAH evaluation), FwPKM consumes stored pairs to update its fast weights.

In our experiments, lookahead targets improved performance and we use them as the default unless otherwise noted.

B.4. Inverse-Distance Weighting (IDW) Scoring

A query–key score in PKM is typically the dot product $s_i = \mathbf{q}^\top K_i$. However, a key row can increase its score by growing its magnitude, without necessarily being *close* to the query in representation space. We therefore consider inverse distance weighting (IDW, McCarter 2023) as a drop-in alternative scoring function:

$$s_i^{\text{IDW}} = -\log\left(\epsilon + \|\mathbf{q} - K_i\|_2^2\right), \tag{20}$$

where we use $\epsilon = 10^{-3}$. Due to the use of Euclidean distance, gradients produced by IDW scores push keys to behave as *prototypes*—centroids of query clusters. We apply IDW by replacing only the query–key scoring function inside PKM's Top-k selection for each sub-key set in product-key retrieval; all other steps of PKM retrieval remain unchanged. Empirically, we found IDW scoring to yield better performance than dot-product scoring in our FwPKM setup.

B.5. Target Normalization and Gradient Clipping

We z-score normalize target values $\mathbf{v}_t$ along the feature dimension to stabilize optimization. Unlike standard slow-weight training, we do not clip gradients for fast-weight updates; this helps FwPKM match the scale of unbounded target values.

B.6. Loss Reduction and Effective Step Size

In Eq. 8, we sum over tokens and features. We avoid mean reduction so that the update magnitude does not shrink with chunk size C or value dimension d_v. Concretely, if one uses mean reduction over (t, j), the gradient of an element $V_{i,j}$ scales by $1/(Cd_v)$, which effectively changes the fast-weight step size.

B.7. Gating-weighted Memorization

We weight each per-token MSE by g_t in Eq. 8. Intuitively, g_t measures how much the model relies on the memory output at this position, so it also serves as a natural importance weight for deciding which tokens should write more strongly. Note that we use gate g_t both to interpolate outputs in Eq. 7 and to weight memorization updates in $\mathcal{L}_{\text{mem}}$ and $\mathcal{L}_{\text{mem}}^{\text{LA}}$.

C. Pseudo Code

Algorithm 1 Fast-weight Product Key Memory (FwPKM) Processing

Input : Hidden states $\mathbf{h}_{1:T} \in \mathbb{R}^{T \times d}$,
Fast weights $\theta = \{V \in \mathbb{R}^{N \times d_v}, K^{(1)}, K^{(2)} \in \mathbb{R}^{\sqrt{N} \times d_k/2}\}$,
Slow weights ϕ

Hyperparameters : Chunk size C, Key dim d_k, Value dim d_v, Top-K k

Output : Output sequence $\mathbf{o}_{1:T} \in \mathbb{R}^{T \times d_v}$

1 **for** $t = 1$ **to** T ***step*** C **do**
   // --- Phase 1: Forward Pass (Inference) ---
2   $\mathbf{h}_{\text{chunk}} \leftarrow \mathbf{h}_{t:t+C}$ // Shape: (C, d)
   // Compute projections using Slow Weights ϕ
3   $\mathbf{q} \leftarrow \texttt{Linear}^q_\phi(\texttt{RMSNorm}^q_\phi(\mathbf{h}_{\text{chunk}}))$ // Shape: (C, d_k)
4   $\mathbf{v} \leftarrow \texttt{Linear}^v_\phi(\texttt{RMSNorm}^v_\phi(\mathbf{h}_{\text{chunk}}))$ // Shape: (C, d_v)
5   $g \leftarrow \sigma\big(\texttt{Linear}^g_\phi(\texttt{RMSNorm}^g_\phi(\mathbf{h}_{\text{chunk}}))\big)$ // Shape: $(C, 1)$
   // Split query into two halves for Product Keys
6   $\mathbf{q}^{(1)}, \mathbf{q}^{(2)} \leftarrow \texttt{Split}(\mathbf{q})$ // Shapes: $(C, d_k/2)$
   // Sparse Retrieval (PKM) with current Fast Weights
7   **for** $m \in \{1, 2\}$ **do**
     // IDW Scoring
8     $s^{(m)} \leftarrow -\log(10^{-3} + ||\mathbf{q}^{(m)} - K^{(m)}||_2^2)$ // Shape: $(C, \sqrt{N})$
     // Top-k Indices
9     $\mathcal{I}^{(m)} \leftarrow \texttt{Top}-\texttt{k}(s^{(m)})$ // Shape: (C, k)
   // Select final Top-k from the $k \times k$ restricted product
10   $S_{\text{target}} \leftarrow \{s_i^{(1)} + s_j^{(2)} \mid (i, j) \in \mathcal{I}^{(1)} \times \mathcal{I}^{(2)}\}$ // k^2 candidates per token
11   $\mathcal{I} \leftarrow \texttt{Top}-\texttt{k}(S_{\text{target}})$ // Final k indices per token
12   $\hat{\mathbf{v}} \leftarrow \sum_{(i,j) \in \mathcal{I}} \texttt{softmax}(S_{\text{target}})_{(i,j)} V_{\texttt{flat-idx}(i,j)}$ // Shape: (C, d_v)
   // Gated Residual Output
13   $\mathbf{o}_{\text{chunk}} \leftarrow g \cdot \hat{\mathbf{v}} + (1 - g) \cdot \mathbf{v}$ // Shape: (C, d_v)

   // --- Phase 2: Backward Pass for Memorization ---
14   $\mathbf{v}_{\text{target}} \leftarrow \texttt{Shift}(\mathbf{v}, +1)$ // Lookahead Target: $\mathbf{v}_{t+1}$
15   $\mathcal{L}^{\text{LA}}_{\text{mem}} \leftarrow \sum_{\text{step}=1}^{C-1} \frac{1}{2} g_{\text{step}} ||\mathbf{v}_{\text{target}} - \hat{\mathbf{v}}_{\text{step}}||_2^2$
16   $\nabla^{\text{agg}}_{V_i} \leftarrow \frac{1}{N_i^{\text{read}}} \nabla_{V_i} \mathcal{L}_{\text{mem}}$ // Aggregate gradients by slot usage
17   $V \leftarrow V - \nabla^{\text{agg}}_V$ // Update Values

   // --- Phase 3: Backward Pass for Addressing (Anti-collapsing) ---
18   **for** $m \in \{1, 2\}$ **do**
19     $\bar{p}^{(m)} \leftarrow \texttt{AvgSlotUsage}(\mathcal{I}^{(m)}, \text{chunk})$ // Shape: $(\sqrt{N},)$
20     $\mathcal{L}^{(m)}_{\text{addr}} \leftarrow -H(\bar{p}^{(m)})$ // Maximize Entropy
21     $K^{(m)} \leftarrow K^{(m)} - \nabla_{K^{(m)}} \mathcal{L}^{(m)}_{\text{addr}}$ // Update Keys
22 **return** $\mathbf{o}_{1:T}$

D. Detailed Experiment Settings

Table 2. Modeling and training hyper-parameters used in the experiments.

Hyperparameter	Value
General	
vocab. size	32000
# layers	12
hidden dim	768
RMS norm ϵ	0.00001
Attention	
head dim	64
# query heads	12
# k/v heads	4
GDN	
conv. size	4
head dim	64
# heads	8
PKM	
key dim	512
value dim	512
# heads	4
Top-K	32
# slots	512^2

Hyperparameter	Value
FwPKM	
key dim	512
value dim	512
# heads	1
Top-K	8
# slots	512^2
chunk size	512
FwMLP	
input dim	512
hidden dim	2304
output dim	512
LR η	0.1
LaCT	
sliding window size	2048
TTT chunk size	512
MoM	
# heads	4
head dim	96
Top-K	2
# experts	4/8/12

Hyperparameter	Value
Training	
max. LR	0.001
min. LR	0.0001
global batch size	128
micro batch size	8
# warmup steps	100
# total steps	20000
weight decay	0.1

Training Every training experiment is conducted on 4 H100 GPUs. The FwPKM module runs under `Float32` precision, other components are converted to `BFloat16` unless specified by the FlashAttention (Dao et al., 2022; Dao, 2024) and the FLA (Yang & Zhang, 2024) kernels.

Evaluation Every evaluation experiment is conducted on 1 H100 GPU. We apply the YaRN method (Peng et al., 2024) to adapt RoPE embeddings to the maximum sequence length in the experiments where sample length exceeds 4096.

Baseline model - `FwMLP` To rule out the impact of implementation details unrelated to the PKM architecture, we propose a baseline model that replaces the PKM in FwPKM with a SwiGLU-MLP that maintains three fast-weight matrices (and their biases) for up, gating, and down projection. The baseline, denoted as `FwMLP`, updates its fast weights by minimizing the MSE loss between its predicted values and target lookahead values at a chunk level. Due to its dense nature, we reduce the MSE losses in a chunk by averaging over both sample and feature dimensions, and we do not apply the loss aggregation and gradient shaping techniques mentioned in Section 2.3. For the same reason, addressing optimization in Section 2.4 is irrelevant too.

Baseline model - `LaCT` We adopt the official implementation of LaCT (Zhang et al., 2025b)[1] as a strong TTT (Sun et al., 2025) baseline. The LaCT architecture consists of a sliding window attention, a fast-weight SwiGLU MLP, and a slow-weight SwiGLU MLP in every layer. The fast weights are optimized to minimize a dot product loss via SGD with momentum or Muon (Jordan et al., 2024), and we found SGD with momentum achieves lower training PPL in our experiments. LaCT uses data-dependent learning rate and L2 weight normalization to improve memorization and retention. We compared several configurations of sliding window size (W) and update chunk size (C). Among $512C + 512W$, $512C + 2048W$, $512C + 4096W$, and $2048C + 2048W$, the best model is $512C + 2048W$.

One notable difference between LaCT (or more generally TTT) and FwMLP/FwPKM is that LaCT/TTT maintains an individual set of fast weights for each sequence in a mini batch, while FwMLP/FwPKM uses a shared set of fast weights for

[1] https://github.com/a1600012888/LaCT/tree/main/lact_llm

all sequences.

Baseline model - `MoM` We use Mixture-of-Memory (Du et al., 2026) as a baseline of sparse fast weights, based on the official FLA implementation. To ensure a fair comparison and a faithful replication of the original paper, we present two MoM variants. The first variant is denoted as `MoM | * Experts` and follows the architecture proposed in the original paper. The second variant, denoted as `* + PKM@6 + MoM@2,10 (* experts)` replaces the original FwPKMs at layer 2 and 10 with an MoM module, directly comparing the performance of an FwPKM layer against an MoM layer.

E. Additional Experimental Results

E.1. Full Perplexity and LongBench Results

In Figure 4, we additionally show the full perplexity results on Fineweb-Edu, LC64, LAMBADA, and five LongBench tasks as bar plots. These results complement the compact main-text table, which focuses on the `GDN+SWA` setting.

E.2. Full Needle-in-a-Haystack Results

We conduct Needle-in-a-Haystack (NIAH, Kamradt 2023; Mohtashami & Jaggi 2023) evaluation to further verify FwPKM's functionality as episodic memory. In all NIAH experiments, we update fast weights after reading the full haystack and after each rereading iteration; we do not perform additional updates while generating the answer.

In the basic setting, we construct 500 NIAH samples from the LAMBADA dataset. Each sample contains a haystack—a 4K-length sequence from the LAMBADA dataset with 5 needles inserted at random positions, where each needle contains a unique 4-character key and a 6-digit value—and a question that requests the value of a specific key. To test FwPKM's large memory storage, we additionally construct test sets of 8K, 32K, and 128K context lengths.

Iterative Memorization (n-iter NIAH). A unique feature of FwPKM is its ability to improve memory fidelity by re-processing the same input. We define an n**-iter NIAH** setting, where the model forwards the same haystack context n times before answering. FwPKM's process chunk size C is accordingly changed to the length of each haystack context such that it updates its memory once after reading an entire haystack. The basic setting is 1-iter NIAH.

Accuracy results of n-iter NIAH ($n \in \{1, 2, 3, 4\}$) on the four test sets of 4K–128K context lengths are shown in Figure 5. The accuracies of all iterations are drawn as a stacked bar.

Finding: Iterative reading boosts retrieval accuracy. For `GDN` and `GDN+SWA` layouts, a single pass (1-iter) is often insufficient for perfect retrieval. However, a second pass (2-iter) yields a massive boost in accuracy, jumping from $< 10\%$ to $> 50\%$ in many cases. More passes further improve the accuracy to $> 70\%$. This confirms that FwPKM effectively exploits test-time training to consolidate episodic memories. In addition, effective iterative memorization correlates with high gating values in Figure 6.

Finding: FwPKM generalizes to 128K contexts. Despite being trained on only 4K-token sequences, FwPKM generalizes effectively to 128K tokens. While `FA` baselines degrade rapidly on context lengths unseen during training, FwPKM maintains robust retrieval performance.

E.3. Gating Analysis

When FwPKM is added to baselines with unrestricted Full Attention (`FA` and `GDN+FA`), perplexity improvements are marginal. Analysis of gating values in Figure 6 reveals that these models learn to ignore FwPKM, with gating weights clustering near zero.

To mitigate this issue, we restrict full attention's long-range perception at *training time* by imposing a sliding attention window of length 512 on all full attention layers with probability 0.9. Denoted by a suffix `pSWA`, these models learn to use FwPKM more, as suggested by increased high-end mass in the gating distributions in Figure 6. While `pSWA` has minimal impact on perplexity, it has a more visible effect in the NIAH evaluation.

E.4. Online Domain Adaptation

We devise an *adapt-and-test* experiment to study how large-scale fast weights adapt to new data domains. From the Pile corpus (Gao et al., 2020), we choose six subsets: *DM Mathematics*, *FreeLaw*, *PhilPapers*, *PubMed Central*, *Ubuntu IRC*, and *USPTO Backgrounds*. These subsets represent different professional domains. For each domain, we sample a 4M-token *adapt set* and a 4M-token *eval set* without overlap. We run 6 adaptation stages sequentially, one per domain's *adapt set*, during which only fast weights are updated. After each adaptation stage, we compute the model's average negative log-likelihood (NLL) on all 6 *eval sets* and report changes relative to the initial NLL before the first stage.

Finding: FwPKM adapts quickly but struggles with retention across stages. As expected, after each adaptation stage, the NLL of the corresponding domain decreases as the fast weights store related domain knowledge. However, these gains often do not persist across subsequent stages, suggesting that previously stored fast-weight knowledge is flushed and replaced by newer domains. This motivates developing a memory retention mechanism to realize long-term continual learning in future work.

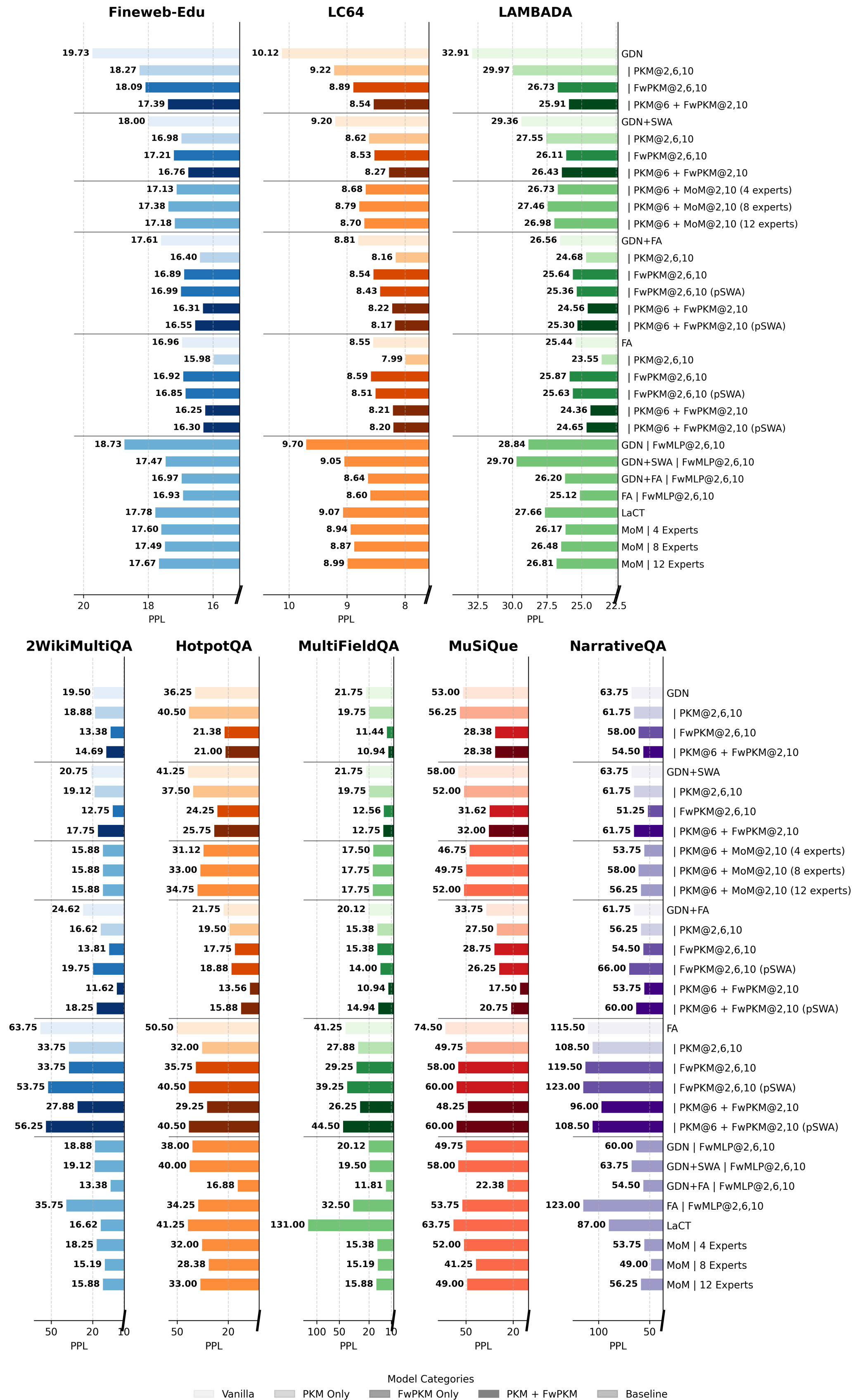

Figure 4. Perplexity evaluation results on Fineweb-Edu, LC64, LAMBADA, and five LongBench tasks.

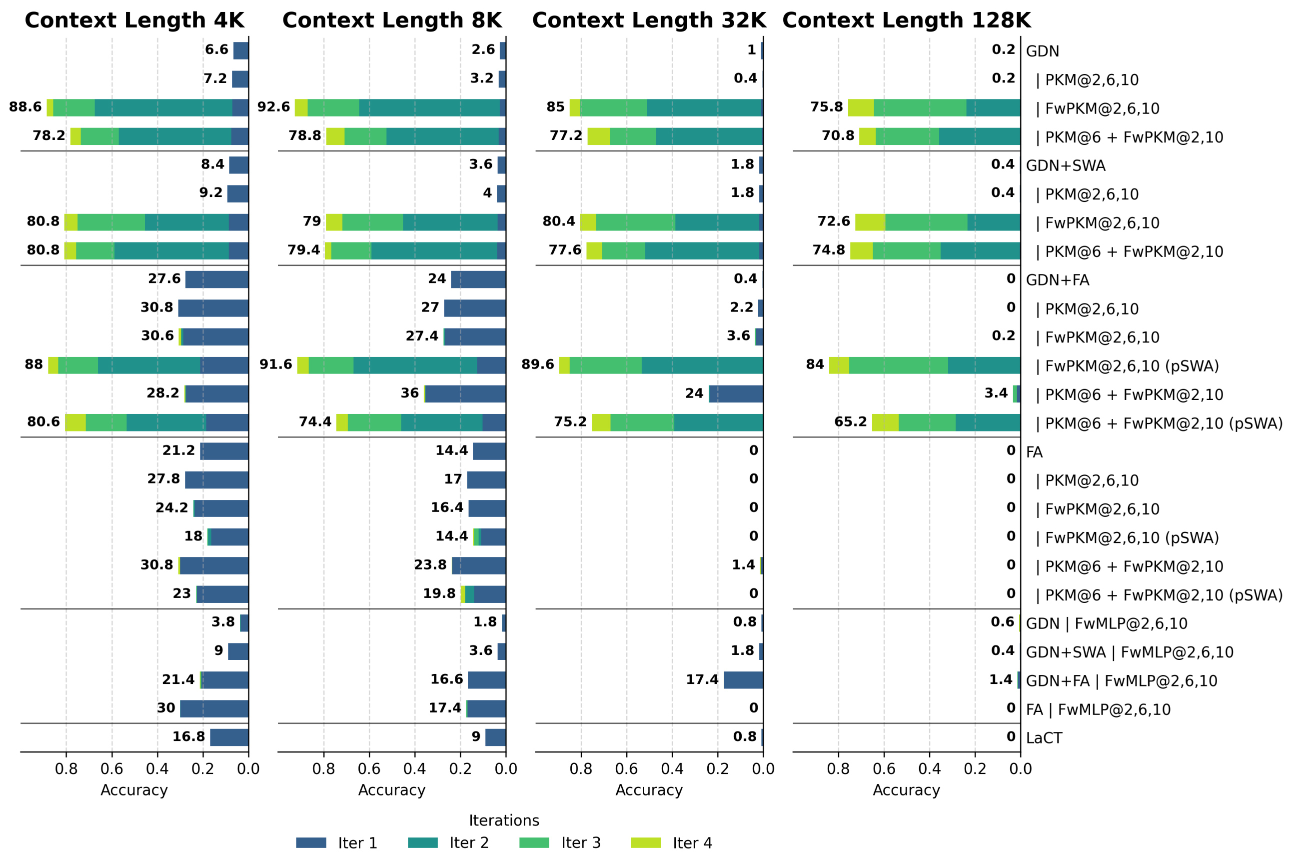

Figure 5. Stacked bar plots for NIAH accuracy results on 4K-/8K-/32K-/128K-length test sets. Each stacked bar shows the accuracies of $\{1, 2, 3, 4\}$-iter NIAH evaluations.

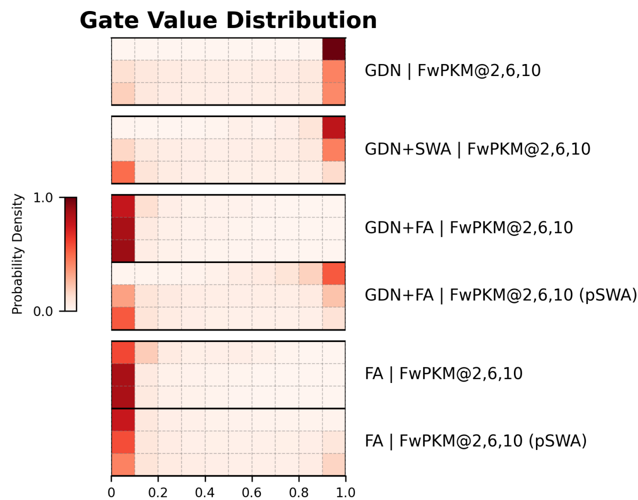

Figure 6. FwPKM gating value distribution. Averaged across the Fineweb-Edu, LC64, and LAMBADA test sets. Each row represents one FwPKM layer. The probability densities are categorized into 10 bins. A darker color suggests a higher probability density.

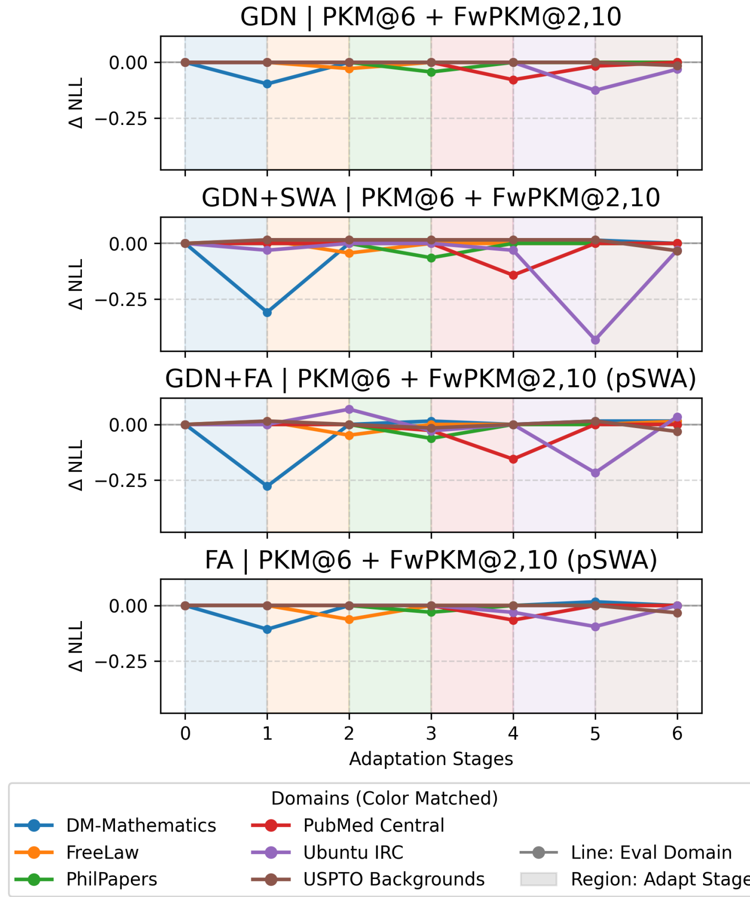

Figure 7. Change of negative log-likelihood (ΔNLL) on six Pile domain datasets before and after 6 adaptation stages of test-time learning on each domain dataset.

F. Cost Analyses

We evaluate the scalability and efficiency of FwPKM by profiling model size and computational costs, summarized in Table 3. Our analysis focuses on three key metrics:

- **Active Parameters**: The subset of parameters actually used during a forward pass, highlighting the architectural sparsity.
- **FLOPs** (Floating Point Operations): The theoretical computational volume required for each step.
- **FLOPS** (FLOPs per Second): The empirical hardware utilization efficiency, derived from actual running time.

Profiling was conducted by running 100 training steps on a single H100 GPU with a global batch size of 8, using the configuration detailed in Section 3 and Appendix D.

Sparsity and Computational Volume Despite a substantial increase in total parameter count (approx. $5\times$ larger than baselines), the **Active Parameter** count of FwPKM models remains comparable to the dense baselines (*e.g.*, `GDN`: 112M vs. `FwPKM`: 116M). Consequently, the theoretical computational cost (**FLOPs**) is remarkably efficient; in some cases, the sparsity of PKM/FwPKM makes them even less FLOPs-intensive than standard dense MLP layers.

Hardware Utilization and Kernels While theoretical costs are low, the wall-clock speed (**Samples per sec.**) and hardware utilization (**FLOPS**) reveal an implementation gap. FwPKM components currently run slower than their dense counterparts. This is primarily due to the maturity difference in kernel optimization: softmax attention and GDN benefit from highly optimized kernels like FlashAttention (Dao et al., 2022; Dao, 2024) and FlashLinearAttention (Yang & Zhang, 2024). In contrast, our sparse operations rely on less specialized implementations. Developing dedicated, hardware-aware kernels for sparse fast-weight updates is a critical future direction to bridge this gap and facilitate broader adoption.

Table 3. Comparison of model size and computation cost. Active parameter counts reflect the sparse layer specifications: PKM utilizes 128 active rows, while FwPKM utilizes 8 active rows per token.

MODEL	Parameters (M)		FLOPs (T)	FLOPS (T/sec.)	Samples/sec.
	Total	Active			
GDN	112.11	112.11	22.04	150.00	54.44
| PKM@2,6,10	509.08	106.62	20.92	112.19	42.90
| FwPKM@2,6,10	519.12	116.48	22.74	70.21	24.70
| PKM@6 + FwPKM@2,10	515.77	113.19	22.14	75.30	27.21
GDN+SWA	112.65	112.65	22.15	151.11	54.58
| PKM@2,6,10	509.61	107.15	21.03	109.43	41.63
| FwPKM@2,6,10	519.66	117.02	22.85	72.55	25.40
| PKM@6 + FwPKM@2,10	516.31	113.73	22.24	75.66	27.21
| Chunk size 128	516.31	113.73	22.24	42.49	15.28
| Chunk size 256	516.31	113.73	22.24	55.47	19.95
| Chunk size 1024	516.31	113.73	22.24	92.24	33.17
| PKM@6 + MoM@2,10 (4 experts)	252.71	116.18	23.17	79.78	27.55
| PKM@6 + MoM@2,10 (8 experts)	257.49	116.18	23.32	74.50	25.56
| PKM@6 + MoM@2,10 (12 experts)	262.26	116.18	23.55	69.43	23.59
| w/o FA & FLA	112.65	112.65	22.15	9.65	3.41
GDN+FA	112.65	112.65	22.16	147.71	53.33
| PKM@2,6,10	509.61	107.15	21.04	102.76	39.07
| FwPKM@2,6,10	519.66	117.02	22.86	69.73	24.40
| PKM@6 + FwPKM@2,10	516.31	113.73	22.25	74.81	26.89
FA	114.25	114.25	22.51	158.75	56.42
| PKM@2,6,10	511.22	108.76	21.39	106.40	39.79
| FwPKM@2,6,10	521.26	118.62	23.21	71.75	24.78
| PKM@6 + FwPKM@2,10	517.91	115.33	22.60	75.71	26.85
GDN | FwMLP@2,6,10	118.05	118.05	23.21	134.04	46.21
GDN+SWA | FwMLP@2,6,10	118.58	118.58	23.31	128.40	44.06
GDN+FA | FwMLP@2,6,10	118.58	118.58	23.32	97.18	33.33
FA | FwMLP@2,6,10	120.19	120.19	23.67	118.85	40.16
LaCT	112.70	112.70	24.44	92.38	30.24
MoM | 4 Experts, 127M	127.69	113.37	24.16	54.80	18.14
MoM | 8 Experts, 156M	156.33	113.37	24.91	36.35	11.67
MoM | 12 Experts, 185M	184.97	113.37	25.13	27.59	8.78

Table 4. Decoding time per token at different prefilling prompt lengths (in ms./token).

Model	4096	16384	32768
GDN	14.6	14.6	14.4
| PKM@2,6,10	19.3	19.5	20.4
| FwPKM@2,6,10	18.5	18.6	18.6
| PKM@6 + FwPKM@2,10	20.1	20.7	18.7
GDN+SWA	14.3	16.7	16.9
| PKM@2,6,10	22.6	23.0	24.7
| FwPKM@2,6,10	21.7	21.8	23.3
| PKM@6 + FwPKM@2,10	21.4	21.5	22.5
| Chunk size 128	19.7	22.4	22.9
| Chunk size 256	23.3	22.6	22.8
| Chunk size 1024	23.1	23.5	25.2
| PKM@6 + MoM@2,10 (4 experts)	39.9	41.2	39.1
| PKM@6 + MoM@2,10 (8 experts)	45.8	50.0	44.6
| PKM@6 + MoM@2,10 (12 experts)	53.1	54.9	53.1
GDN+FA	15.7	16.2	17.0
| PKM@2,6,10	21.7	23.3	23.7
| FwPKM@2,6,10	20.6	23.2	24.8
| PKM@6 + FwPKM@2,10	22.5	24.9	25.2
FA	14.2	18.4	21.5
| PKM@2,6,10	26.6	27.3	30.0
| FwPKM@2,6,10	24.6	27.5	27.6
| PKM@6 + FwPKM@2,10	15.4	25.8	26.6
GDN | FwMLP@2,6,10	14.2	14.4	14.4
GDN+SWA | FwMLP@2,6,10	16.1	16.8	16.7
GDN+FA | FwMLP@2,6,10	15.9	16.2	16.9
FA | FwMLP@2,6,10	17.3	19.1	21.8
LaCT	39.3	39.5	39.4
MoM | 4 Experts, 127M	148.8	142.9	144.0
MoM | 8 Experts, 156M	200.9	200.2	198.5
MoM | 12 Experts, 185M	270.8	269.1	271.9

Table 5. Ablation study: Comparison of model size and computation cost.

MODEL	Parameters (M)		FLOPs (T)	FLOPS (T/sec.)	Samples/sec.
	Total	Active			
`GDN+SWA	PKM@6 + FwPKM@2,10`	516.31	113.73	22.24	58.55	21.06
`| w/ 1 head x Top-32`	516.31	113.76	22.24	58.28	20.96
`| w/ 4 heads x Top-8`	520.24	117.69	22.71	48.47	17.08
`| w/o value norm`	516.31	113.73	22.24	65.03	23.39
`| w/o addr loss`	516.31	113.73	22.24	65.91	23.70
`| w/o gating`	516.30	113.72	22.24	61.30	22.05
`| w/o loss weight`	516.31	113.73	22.24	61.06	21.96
`| w/o lookahead`	516.31	113.73	22.24	58.70	21.11

G. Ablation Study

To understand the influence of techniques proposed in Section 2, we conduct ablation experiments based on the `GDN+SWA | PKM6+FwPKM@2,10` model. The following variants are trained and evaluated using the same pipelines.

- "`w/ 1 head × Top-32`" uses a different Top-K setting as the name suggests.
- "`w/ 4 heads × Top-8`" uses a different multi-head setting as the name suggests.
- "`w/o value norm`" does NOT z-score normalize target values.
- "`w/o addr loss`" does NOT use the marginal entropy loss to update key matrices, instead it uses the MSE loss to update both the keys and value matrices.
- "`w/o gating`" does NOT use g_t.
- "`w/o loss weight`" uses g_t but does NOT weight MSE loss with it.
- "`w/o lookahead`" does NOT use lookahead values as MSE targets.

In addition, we found that replacing IDW score with dot-product score often results in memory collapsing and loss divergence so do not include it in the ablation study.

G.1. Evaluation

As shown in Figures 8, 9, 10, 11, and 12, we conduct the same evaluation experiments from Section 3 to ablated models. a) Notably, removing lookahead values yields the most significant harm to model performance. Many techniques bring slight PPL improvement, but lead to less healthy memory utility and subsequently worse NIAH accuracies to different extents. b) Addressing loss and weighting MSE loss with gating values are important for NIAH retrieval accuracy. c) A larger effective top-k slightly improves performance in some tasks, but the default setting of 1 head $\times$ 8 slots strikes a sweet spot between task performance and compute efficiency (See computation cost of ablated models in Table 5).

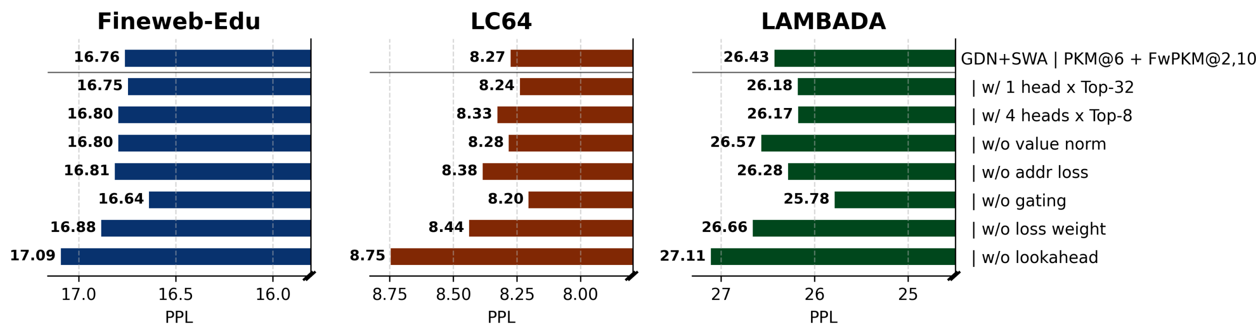

Figure 8. Ablation study: Perplexity on Fineweb-Edu, LC64, and LAMBADA. We use bar colors to help distinguish between models with different types of memory components.

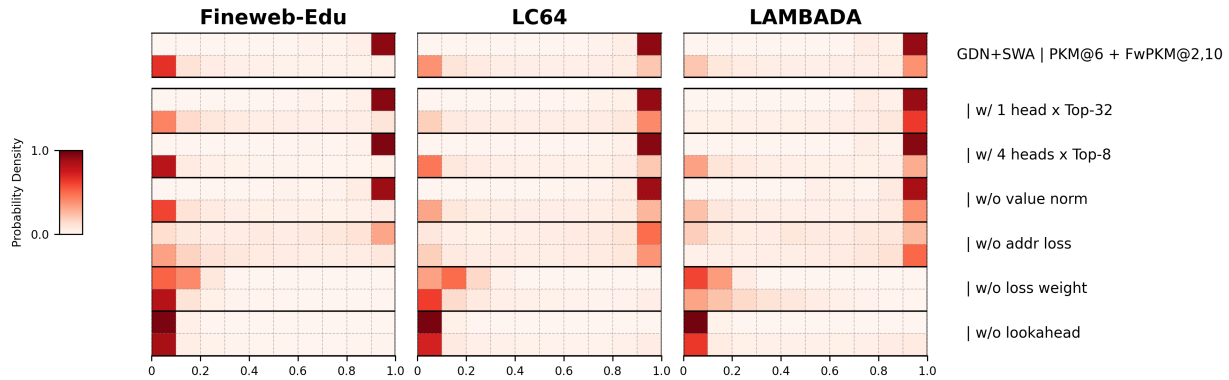

Figure 9. Ablation study: FwPKM gating value distribution on Fineweb-Edu, LC64, and LAMBADA test sets. Each row represents one FwPKM layer. The probability densities are categorized into 10 bins. A darker color suggests a higher probability density of a bin.

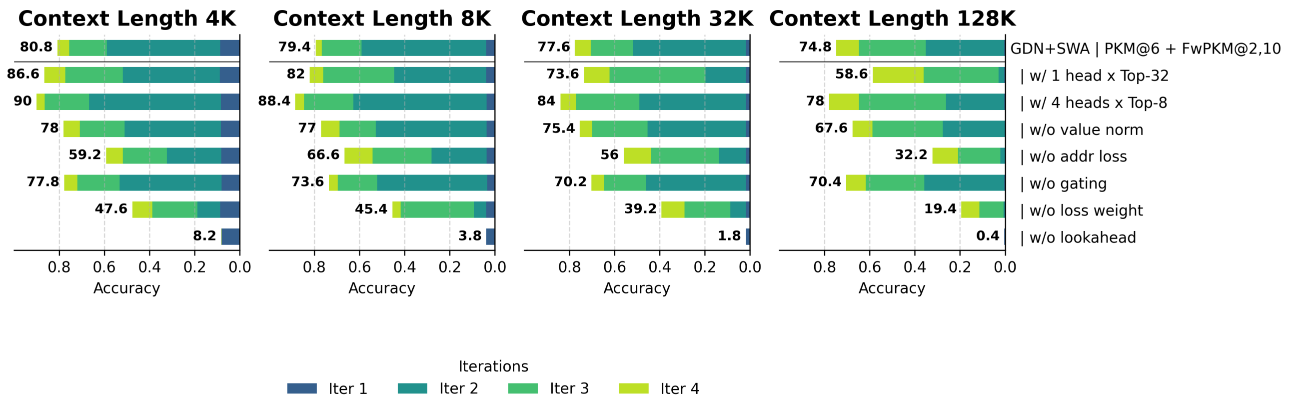

Figure 10. Ablation study: Stacked bar plots for NIAH accuracy results on 4K-/8K-/32K-/128K-length test sets. Each stacked bar shows the accuracies of $\{1, 2, 3, 4\}$-iter NIAH evaluations.

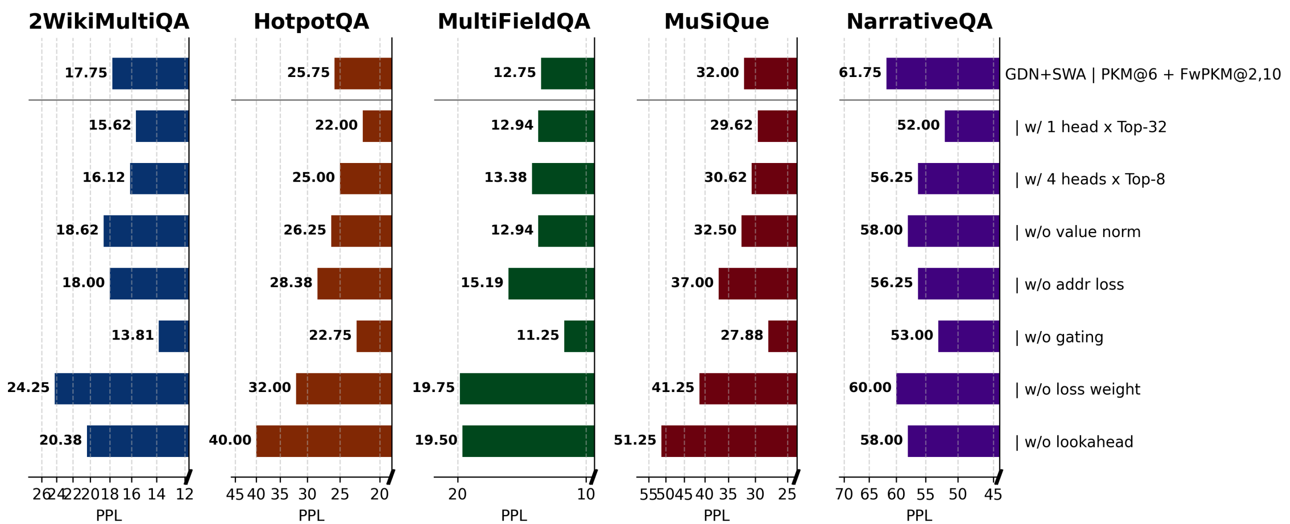

Figure 11. Ablation study: Perplexity on five Longbench tasks. We use bar colors to help distinguish between models with different types of memory components.

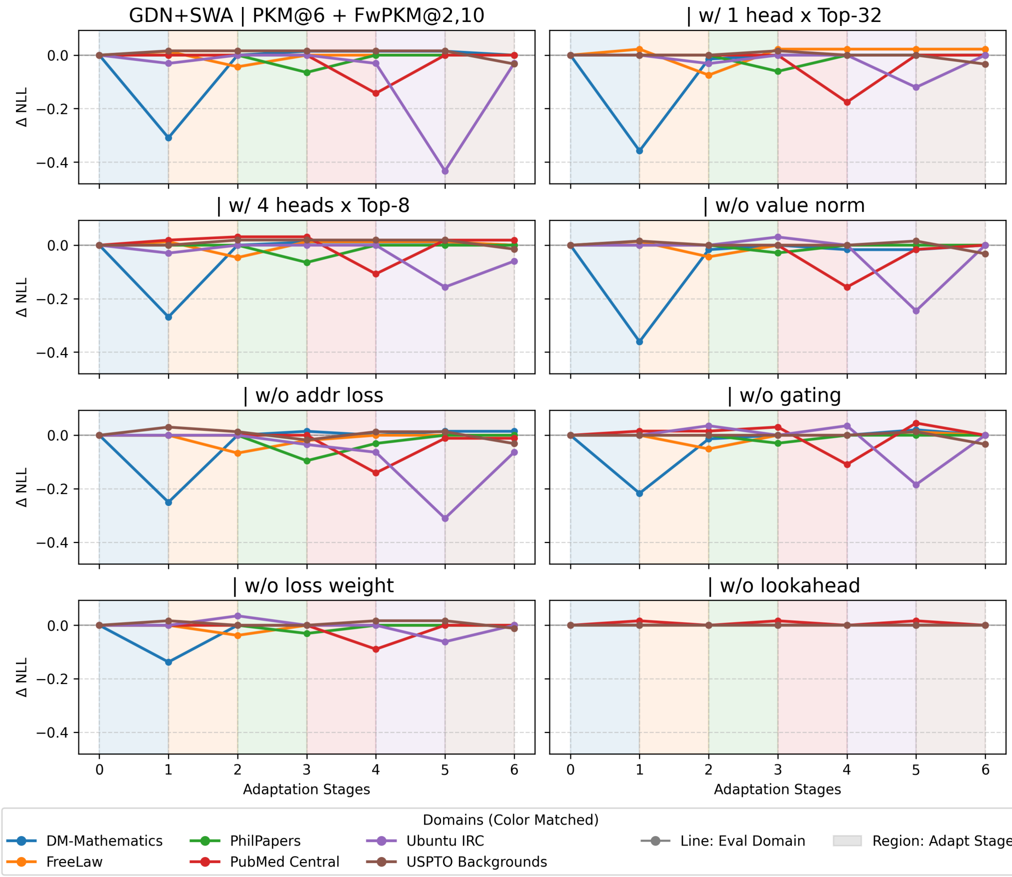

Figure 12. Ablation study: Change of negative log-likelihood (ΔNLL) on six Pile domain datasets before and after 6 adaptation stages of test-time learning on each domain dataset.

G.2. Freezing Fast Weights

To consolidate the evidence that FwPKM contributes to long-context language modeling, we conduct extra PPL evaluation experiments as in Section 3 but *freezing* FwPKM's fast-weight parameters (*i.e.* value matrix V and key matrices $K^{(1)}, K^{(2)}$). Figure 13 shows that freezing fast weights significantly increase perplexity on long-context datasets LC64 and LAMBADA while Fineweb-Edu is less affected, further supporting conclusions in Section 3.

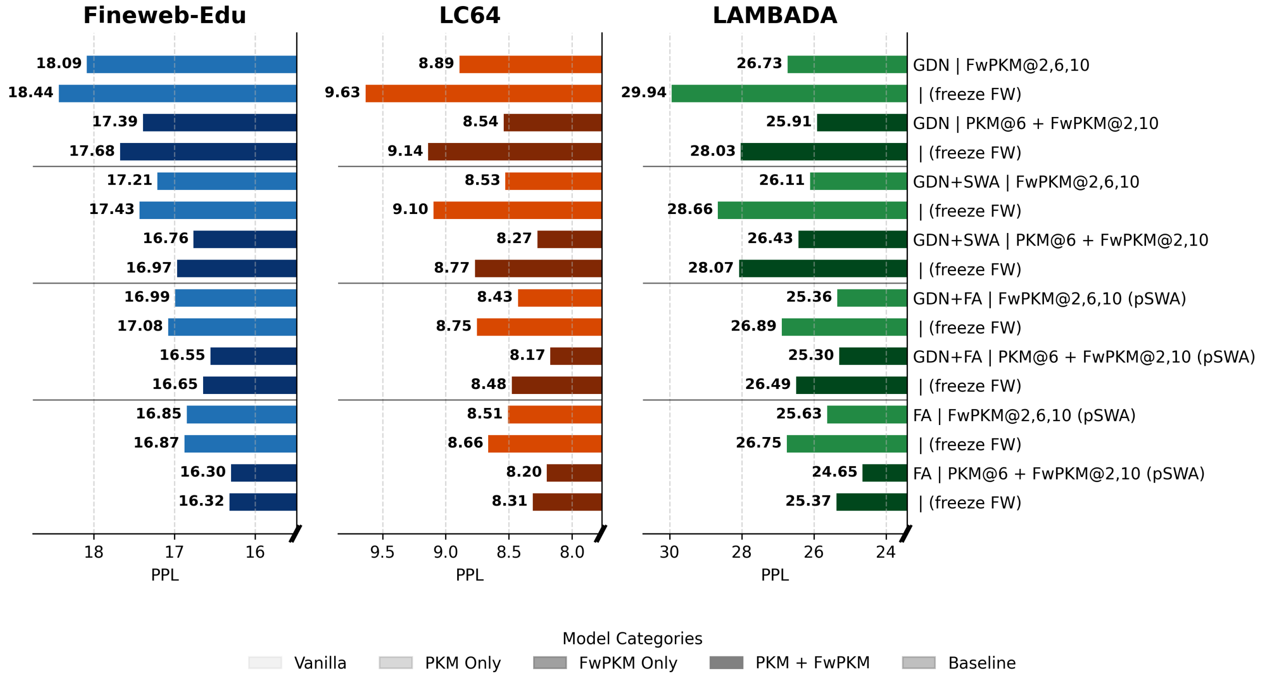

Figure 13. Ablation study: Perplexity on Fineweb-Edu, LC64, and LAMBADA. Unlike previous experiments, the fast-weight parameters in FwPKM are frozen. We use bar colors to help distinguish between models with different types of memory components.

G.3. Addressing Metrics

The design choices have different impact on the evenness of slot usage, and we use three addressing metrics to characterize it. Recall that we define N_i^{read} to be the "row contribution", *i.e.* the number of times a value row V_i is accessed.

- **Collision rate** is the ratio of *competitive* slot accesses: $\sum_{N_i^{\text{read}}>1} N_i^{\text{read}}/N^{\text{total}}$, where N^{total} is the total number of slot accesses.
- **Coverage rate** is the ratio of accessed slots: $\sum_{N_i^{\text{read}}>0} 1$ / N.
- **KLD** is the Kullback–Leibler divergence between the average slot use distribution and an uniform distribution:

$$p_i^{\text{read}} = N_i^{\text{read}} / \sum_i^N N_i^{\text{read}}$$

$$\text{KLD} = \sum_i^N p_i^{\text{read}} \log \frac{p_i^{\text{read}}}{1/N}.$$

For each evaluation dataset and each metric, we calculate the metric every 32K tokens, which gives us 8M / 32K = 256 numbers and we take their average. We choose 32K tokens because a `GDN+SWA | PKM@6+FwPKM@2,10` model selects 8 slots per token and perfectly balanced slot selection would cover 32K×8=256K slots, which is the exact number of FwPKM slots. We report the metrics averaged across two FwPKM layers.

As shown in Table 6, memory slot use is more balanced on long-context datasets LAMBADA and LC64 but still far from uniform distribution. Increasing head number or Top-K can improve the evenness but also increases collision rate. Ablating the addressing loss leads to a concentrated slot access pattern.

Unhealthy slot access statistics such as in the "`w/o addr loss`" model are paired with degraded performance in certain tasks (*e.g.* NIAH). If we compare models of effectively Top-32 slot activations (*i.e.* "`w/ 1 head x Top-32`" and "`w/ 4 heads x Top-8`") against models of effectively Top-8 slot activations, despite the former group having higher coverage and lower KLD, their differences in evaluation experiments (Figures 8, 9, 10, 11, and 12) do not correlate with the difference in slot access statistics. The results might be affected by compound factors such as the Top-k mechanism per se, but we believe it is important to mitigate severely unbalanced memory slot access, for example by minimizing the addressing loss.

Table 6. Memory addressing metrics. Coll. – collision rate (%). Cov. – coverage rate (%). KLD – KLD against uniform distribution. Darker colors indicate better performance (*i.e.* low collision, high coverage, low KLD).

Model	**Fineweb-Edu**			**LAMBADA**			**LC64**		
	Coll. ↓	Cov. ↑	KLD ↓	Coll .↓	Cov .↑	KLD ↓	Coll .↓	Cov .↑	KLD ↓
`GDN+SWA | PKM@6 + FwPKM@2,10`	55.7	15.8	2.2	83.1	32.1	1.9	85.2	28.4	2.1
`| w/ 1 head x Top-32`	78.1	41.7	1.3	95.3	64.1	1.3	95.4	55.0	1.9
`| w/ 4 heads x Top-8`	78.2	41.4	1.4	95.5	60.8	1.4	95.3	60.9	1.7
`| w/o value norm`	55.6	15.8	2.2	83.1	32.1	1.8	84.9	28.9	2.1
`| w/o addr loss`	84.6	7.2	3.6	94.9	12.0	3.9	96.4	8.0	5.4
`| w/o gating`	55.2	16.0	2.2	83.1	32.1	1.9	84.3	29.9	2.0
`| w/o loss weight`	57.3	15.5	2.2	83.8	31.2	1.9	85.6	28.1	2.1
`| w/o lookahead`	52.5	16.6	2.1	82.4	33.1	1.8	82.5	32.8	1.8

H. Interpretability Analyses

A key advantage of FwPKM over black-box architectures is its inherent interpretability. Because memory slots are explicitly written and read, we can trace retrieved information back to specific input tokens.

H.1. Case Study: Probing Memory in NIAH

We investigate the internal retrieval mechanism by visualizing the memory access patterns of the `GDN+SWA|PKM@6+FwPKM@2,10` model during a 4-iter Needle-in-a-Haystack (NIAH) task with a 4K context. Specifically, we trace the model's internal state as it generates the 6-digit passkey answer.

Figure 14 displays the Top-8 retrieved memory slots for both a lower layer (Layer 2) and a higher layer (Layer 10) at each generation step. For every retrieved slot, we decode and display the content stored during the latest memorization step: the query token (the key used for addressing), the target value token (the predicted next token), and the surrounding context (the text window surrounding the target in the original haystack). To facilitate analysis, we mark slots with a `[HIT]` label if they explicitly store the ground-truth next token required for the current generation step.

High-precision retrieval As shown in Figure 14, the majority of retrieved slots contain the correct target tokens associated with the query needle. The model successfully surfaces the specific needle (ID "8320" at depth 89.66%) from thousands of tokens, populating the top slots with the correct 6-digit value sequence ("964876").

Robust aggregation Despite individual slot errors, the model successfully aggregates information across the 16 slots from the two FwPKM layers to generate the correct 6-digit value. This demonstrates that FwPKM functions as a robust distributed storage system, where consensus across multiple slots compensates for noise in individual retrievals.

H.2. Case Study: Selective Gating

We further examine the "gating values" g_t to understand when the model chooses to rely on episodic memory versus its static slow weights. To visualize this, we feed a introduction to this paper into the network. We adjust the update chunk size from 512 to 32 to capture fine-grained local adaptivity.

Figure 15 presents a heatmap of the gating scalar g_t with input text in the foreground. Higher intensity (represented by darker bars) indicates a value of g_t closer to 1, signifying a strong reliance on the fast-weight episodic memory. Conversely, lower intensity indicates reliance on the pre-trained slow weights.

Layer specialization We observe distinct behaviors across model depths. The lower-layer FwPKM tends to maintain high gating values across almost all tokens, acting as a general-purpose buffer that indiscriminately caches recent history. In contrast, the higher-layer FwPKM exhibits highly selective activation.

Novelty detection As seen in Figure 15 (layer 10), gating values spike specifically for tokens that are out-of-domain regarding the model's training dataset (*e.g.* the citation tokens). This indicates that the model effectively distinguishes between general linguistic patterns (processed by slow weights) and novel, context-specific entities (processed by fast weights).

Needles

... ID-8320: 964876 ... ID-d83f: 645749 ... ID-97d9: 831209 ... ID-3ba3: 158393 ... ID-8fab: 099170 ...

Output Token 1: '9'

--- FwPKM Layer 2 ---

```
Slot 161817 | [HIT] | Q:      | V in ctx:...D-8320 is 964876. cry...
Slot 162216 | [HIT] | Q:      | V in ctx:...D-8320 is 964876. cry...
Slot  14361 | [HIT] | Q:      | V in ctx:...D-8320 is 964876. cry...
Slot 161945 | [HIT] | Q:      | V in ctx:...D-8320 is 964876. cry...
Slot  14760 | [HIT] | Q:      | V in ctx:...D-8320 is 964876. cry...
Slot  28697 | [HIT] | Q:      | V in ctx:...D-8320 is 964876. cry...
Slot 129049 |       | Q: hand | V in ctx:...g one hand on top of th...
Slot 162198 | [HIT] | Q:      | V in ctx:...D-8320 is 964876. cry...
```

--- FwPKM Layer 10 ---

```
Slot   1506 |       | Q:      | V in ctx:...D-97d9 is 831209. a f...
Slot   1064 |       | Q:      | V in ctx:...D-97d9 is 831209. a f...
Slot 148962 |       | Q:the   | V in ctx:...pie-in-the-sky while ...
Slot 148520 |       | Q: bottl| V in ctx:...f a bottle of water . s...
Slot  24034 |       | Q: membe| V in ctx:...ior member of the leade...
Slot 234978 |       | Q:      | V in ctx:...D-4ad4 is 512407. nev...
Slot 175074 |       | Q:using | V in ctx:...n accusing voice bit back ...
Slot 108514 |       | Q:      | V in ctx:...D-7ff8 is 930401. set...
```

Output Token 2: '6'

--- FwPKM Layer 2 ---

```
Slot  26757 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
Slot  28805 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
Slot  27040 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
Slot  26781 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
Slot 159877 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
Slot  27125 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
Slot  26899 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
Slot 135301 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
```

--- FwPKM Layer 10 ---

```
Slot 172940 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
Slot 235404 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
Slot 216972 |       | Q:5     | V in ctx:...ffb2 is 253297. throu...
Slot 222092 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
Slot  51084 |       | Q: now  | V in ctx:...i have now is there in ...
Slot  94092 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
Slot  28556 |       | Q:0     | V in ctx:...-cd05 is 075065. . no...
Slot  29068 | [HIT] | Q:9     | V in ctx:...-8320 is 964876. cryi...
```

Output Token 3: '4'

--- FwPKM Layer 2 ---

```
Slot 121730 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
Slot  21378 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
Slot  54146 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
Slot 105858 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
Slot  63874 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
Slot 164738 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
Slot 121524 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
Slot  49538 |       (Empty)
```

--- FwPKM Layer 10 ---

```
Slot 139652 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
Slot 139398 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
Slot  98692 |       | Q: .    | V in ctx:...ttention . there were a bu...
Slot 139584 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
Slot 139747 |       | Q:ups   | V in ctx:...of pickups . i should ...
Slot 139358 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
Slot 139479 |       | Q:4     | V in ctx:...4884 is 543884. shara...
Slot 139690 | [HIT] | Q:6     | V in ctx:...8320 is 964876. cryin...
```

Output Token 4: '8'

--- FwPKM Layer 2 ---

```
Slot  39146 |       | Q: frown| V in ctx:...en frowned at the page ...
Slot 103146 | [HIT] | Q:4     | V in ctx:...320 is 964876. crying...
Slot  38939 | [HIT] | Q:4     | V in ctx:...320 is 964876. crying...
Slot  39022 | [HIT] | Q:4     | V in ctx:...320 is 964876. crying...
Slot 142058 | [HIT] | Q:4     | V in ctx:...320 is 964876. crying...
Slot  39388 | [HIT] | Q:4     | V in ctx:...320 is 964876. crying...
Slot  39266 | [HIT] | Q:4     | V in ctx:...320 is 964876. crying...
Slot 102939 | [HIT] | Q:4     | V in ctx:...320 is 964876. crying...
```

--- FwPKM Layer 10 ---

```
Slot 249564 | [HIT] | Q:4     | V in ctx:...320 is 964876. crying...
Slot 249364 | [HIT] | Q:4     | V in ctx:...320 is 964876. crying...
Slot 153820 |       | Q:3     | V in ctx:...      82382bed wall ...
Slot 249607 |       | Q:8     | V in ctx:...18 is 183863. going n...
Slot 153620 | [HIT] | Q:4     | V in ctx:...320 is 964876. crying...
Slot 153863 |       | Q:1     | V in ctx:...4b is 859199. you wan...
Slot 249413 |       | Q: next | V in ctx:...right next to me , and ...
Slot 249723 |       (Empty)
```

Output Token 5: '7'

--- FwPKM Layer 2 ---

```
Slot 209912 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
Slot 104440 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
Slot 215544 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
Slot 209550 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
Slot 190968 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
Slot 104078 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
Slot 215182 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
Slot  49144 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
```

--- FwPKM Layer 10 ---

```
Slot  55559 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
Slot  55515 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
Slot  55603 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
Slot 164103 | [HIT] | Q:8     | V in ctx:...20 is 964876. crying ...
Slot  55782 |       | Q: run  | V in ctx:...ted to run away from here...
Slot  71431 |       | Q:8     | V in ctx:...4 is 818180. beloved ...
Slot  55511 |       (Empty)
Slot 164059 |       | Q:9     | V in ctx:...b is 859199. you want...
```

Output Token 6: '6'

--- FwPKM Layer 2 ---

```
Slot 161835 | [HIT] | Q:7     | V in ctx:...0 is 964876. crying ....
Slot  82475 | [HIT] | Q:7     | V in ctx:...0 is 964876. crying ....
Slot 180779 | [HIT] | Q:7     | V in ctx:...0 is 964876. crying ....
Slot  13867 | [HIT] | Q:7     | V in ctx:...0 is 964876. crying ....
Slot 166443 | [HIT] | Q:7     | V in ctx:...0 is 964876. crying ....
Slot 158251 | [HIT] | Q:7     | V in ctx:...0 is 964876. crying ....
Slot 222251 | [HIT] | Q:7     | V in ctx:...0 is 964876. crying ....
Slot  56875 |       | Q:7     | V in ctx:... is 253297. through h...
```

--- FwPKM Layer 10 ---

```
Slot 141703 |       | Q:ding  | V in ctx:...s thudding against her ribs ...
Slot 172935 |       | Q: guys | V in ctx:...erous guys . wara froz...
Slot  54663 | [HIT] | Q:7     | V in ctx:...0 is 964876. crying ....
Slot 157063 |       | Q: alive| V in ctx:...till alive . before th...
Slot 141517 |       | Q:3     | V in ctx:... is 158393. his legs ...
Slot 172749 | [HIT] | Q:7     | V in ctx:...0 is 964876. crying ....
Slot  54477 | [HIT] | Q:7     | V in ctx:...0 is 964876. crying ....
Slot  71047 |       | Q: .    | V in ctx:...is place . i go here a...
```

Figure 14. An example of FwPKM slot access of GDN+SWA|PKM@6+FwPKM@2,10 during generating an NIAH-4K answer. The model memorizes the haystack for 3 extra iterations, *i.e.* 4-iter NIAH. See Section H.1 for the visual elements' definitions.

Long-context language modeling often requires remembering specific, local facts (e.g., a name, a variable binding, a constraint) introduced tens of thousands of tokens earlier, and retrieving them when needed. The core sequence modeling layers, or token mixers, can be viewed as associative memory systems (Behrouz et al., 2025c; Dao and Gu, 2024; Gu and Dao, 2024; Peng et al., 2025; Sun et al., 2025; Vaswani et al., 2017; Yang et al., 2024b, 2025). In this framework, information from past tokens is encoded into storage (i.e., the process of memorization) and later integrated into a prediction step by querying that storage (i.e., the process of retrieval).\n\nThe current landscape is defined by a trade-off between memory capacity and computational efficiency. Softmax self-attention (Vaswani et al., 2017; Zhong et al., 2025) memorizes by explicitly storing a key-value pair for every past token and retrieves by comparing the current query to all stored keys, producing a weighted sum of values. This yields high-fidelity retrieval but at quadratic cost with sequence length. In contrast, modern Recurrent Neural Networks (RNNs) such as linear attention (Katharopoulos et al., 2020), Mamba2 (Dao and Gu, 2024), and DeltaNet (Schlag et al., 2021a; Yang et al., 2024b) compress history into a fixed-size state using learned update rules (e.g., Hebbian-style accumulation or Delta-rule error correction) and retrieve via a state readout in the form of vector-matrix multiplication. While this enables efficient, sub-quadratic retrieval, the fixed state capacity inherently limits the depth of information the model can retain compared to softmax attention.\n\nTo address the capacity limitations of these modern RNNs, recent work on Test-Time Training (TTT, Sun et al., 2025; Zhang et al., 2025b) and Titans (Behrouz et al., 2025c) replaces the fixed state matrix with a "fast-weight" (Hinton and Plaut, 1987; Schmidhuber, 1992) neural network (typically an MLP). Similar to a state matrix, this MLP adapts to input sequences on-the-fly by minimizing a reconstruction loss. However, these approaches face a scaling bottleneck: dense network structure. An MLP's computation scales linearly with its parameter count. To store the massive amount of information required for long-context tasks, the fast-weight MLP must be large; yet, updating and querying a large, dense MLP frequently is computationally infeasible.\n

Figure 15. GDN+SWA|PKM@6+FwPKM@2,10's FwPKM gating values on tokens from an introduction paragraph. The blue color in the background denotes the gating intensity of the FwPKM at layer 2 and the red color denotes the FwPKM at layer 10. A darker color represents higher intensity. Black vertical lines show positions where FwPKM's fast weights are updated, which is every 32 tokens as we specified for this visualization example.

I. Improving Perplexity over Sequence Boundary

In long-context datasets LC64 and LAMBADA, a document can span over multiple 4K-token sequences. The ability of carrying over memory states across sequence boundaries makes FwPKM models excel at reducing the PPL of tokens that require cross-sequence dependency.

In Figure 16, we demonstrate the advantage of FwPKM models (`*|PKM@6+FwPKM@2,10`) by showing their PPL reduction from vanilla counterparts (`GDN`, `GDN+SWA`, `GDN+FA`) on the first ten LC64 documents used in Section 3. Within each document, we calculate the average of PPLs cumulated from the beginning of the document. Then we plot the reduction of cumulative PPL every 128 tokens. The orange lines denote the cumulative PPL difference between "`*|PKM@6+FwPKM@2,10`" and their vanilla counterparts, and the blue lines show the PPL difference between "`*|PKM@2,6,10`" and vanilla counterparts as reference.

While the PKM-only variants show significant PPL reduction over the vanilla models, the PPL reduction curves are relatively smooth, suggesting constant advantage throughout each document. On the other hand, the FwPKM models have more jagged curves. We hypothesize that these "PPL drops" are caused by FwPKM exploiting information from previous sequences, which is impossible for the vanilla and PKM-only models.

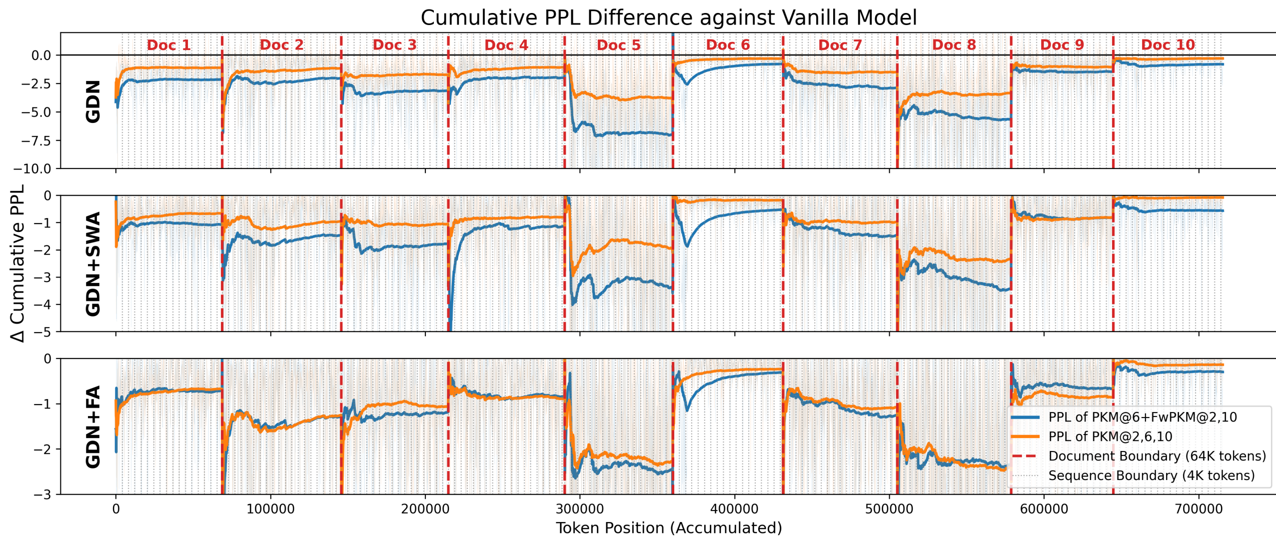

Figure 16. PPL reduction of "`*|PKM@6+FwPKM@2,10`" and "`*|PKM@2,6,10`" from vanilla `GDN`, `GDN+SWA`, and `GDN+FA` models at different token positions.