MIXTURE OF CHAPTERS: SCALING LEARNT MEMORY IN TRANSFORMERS

Tasmay Pankaj Tibrewal[1,2] **Pritish Saha**[1] **Ankit Meda**[1]
Kunal Singh[2] **Pradeep Moturi**[2]

[1]IIT Kharagpur [2]Fractal AI

tasmay.tibrewal@kgpian.iitkgp.ac.in, pritish.saha@kgpian.iitkgp.ac.in,
ankitm18@kgpian.iitkgp.ac.in, kunal.singh@fractal.ai,
pradeep.moturi@fractal.ai

ABSTRACT

Transformers lack an explicit architectural mechanism for storing and organizing knowledge acquired during training. We introduce learnable sparse memory banks: a set of latent tokens, randomly initialized and trained end-to-end, that transformer layers query via cross-attention to retrieve stored knowledge. To scale memory capacity without prohibitive attention costs, we propose chapter-based routing inspired by Mixture-of-Experts architectures, partitioning the memory bank into chapters and training a router to select relevant subsets per input. This enables scaling to 262K memory tokens while maintaining tractable computation. We evaluate our approach against standard transformers (in iso-FLOP settings) on pre-training and instruction fine-tuning across relevant benchmarks. Our models surpass iso-FLOP baselines suggesting scope for a new axis of scaling, demonstrating that explicit associative memory provides complementary capacity to what is captured implicitly in model parameters. Additionally, we observe improved knowledge retention under continued training, with robustness to forgetting when transitioning between training phases (e.g., pretraining to instruction fine-tuning).

1 INTRODUCTION

Transformers provide a powerful foundation for sequence modeling (Vaswani et al., 2017), but they do not expose an explicit architectural mechanism for persistent, addressable memory. In practice, many systems instead rely on inference-time mechanisms such as segment-level recurrence over prior hidden states (Dai et al., 2019a), KV-cache retention, eviction, or compression methods for long-context generation (Xiao et al., 2024; Zhang et al., 2023b; Chen et al., 2024; Karami et al., 2025), tool-based updates (Schick et al., 2023; Yao et al., 2023), and retrieval-augmented generation (RAG) (Lewis et al., 2020b). These methods are effective, but they primarily reuse or manage previously processed context, or retrieve external information, rather than providing a learned internal memory store for factual knowledge acquired during training.

We study a complementary direction: a transformer augmented with a learned memory bank (Wu et al., 2020; Sukhbaatar et al., 2019b). The memory is a set of latent tokens, randomly initialized and trained end-to-end, that can be queried via cross-attention, where the token stream produces queries and the memory provides keys and values (Wu et al., 2020). This design stores knowledge in a compact latent space rather than as retrieved text, and it can be integrated inside standard transformer blocks (Wu et al., 2020; Jaegle et al., 2021a).

The main bottleneck is scale. Dense cross-attention over a very large bank is expensive. We therefore introduce **Mixture of Chapters (MoC)**[1], which partitions the memory bank into chapters and uses a lightweight router to select a small subset of chapters per input at the sequence level (Shazeer et al., 2017; Fedus et al., 2022; Roy et al., 2021). This enables scaling learned memory capacity to large

[1]Code is available at https://github.com/Tasmay-Tibrewal/Memory.

sizes while keeping computation tractable, and it fits naturally with the goals of associative memory research (Berges et al., 2025; Lample et al., 2019b).

2 MOTIVATION

We motivate **Mixture of Chapters** with three goals.

(1) Explicit memory. Transformers store knowledge implicitly in dense parameters (Vaswani et al., 2017), making it hard to inspect, edit, or scale memory independently. Prior work adds explicit memory via large key-value tables (Lample et al., 2019a; Berges et al., 2024) or attention-based memory modules (Wu et al., 2020; de Jong et al., 2021b). We follow this direction by introducing a learned, addressable memory bank that complements parametric capacity.

(2) Scalable sparse access. Dense attention over large memory banks is expensive (Lample et al., 2019a). Inspired by sparse routing in MoE models (Shazeer et al., 2017; Fedus et al., 2021) and routing-based sparsification (Roy et al., 2020), we partition memory into chapters and route each input to a small subset, enabling predictable scaling at tractable cost.

(3) Retention under continued training. Post-training shifts can cause catastrophic forgetting (Kirkpatrick et al., 2017), including in LLM instruction tuning (Luo et al., 2023). An explicit memory bank can reduce interference during continued training by anchoring factual content, which we observe empirically across training phase transitions (Cossu et al., 2022).

3 RELATED WORK

Most relevant. Our work is closest to learned, scalable internal memory modules integrated into transformers. Product Key Memory (PKM) provides a classic recipe for large trainable key-value memory with efficient sparse lookup (Lample et al., 2019a). Memory Layers at Scale shows that trainable memory layers can add substantial capacity with near-constant FLOPs and strong factual gains (Berges et al., 2025). We build on this direction, but use a learned latent-token memory bank accessed via cross-attention and scale it with sequence-level chapter routing.

Other memory mechanisms. A broad set of alternatives externalize memory at inference time or focus on efficiency and long-context handling. Transformer-XL exemplifies recurrent caching to extend effective context (Dai et al., 2019b), and RAG exemplifies retrieval over an external corpus (Lewis et al., 2020a). Conditional computation via routing, as in mixture-of-experts, provides the sparse-activation template we adapt for memory selection (Shazeer et al., 2017). Finally, continual training can induce catastrophic forgetting (Kirkpatrick et al., 2017), which motivates our retention experiments. A wider map of related directions is provided in Appendix A.1 (Table 3).

4 CONTRIBUTIONS

This paper makes the following contributions:

- **Learned associative memory bank in transformers.** We add a persistent bank of latent memory tokens trained end-to-end and accessed via cross-attention, enabling internal retrieval in a compact latent space rather than text-based retrieval (Wu et al., 2020; Berges et al., 2024).
- **Mixture of Chapters for scalable sparse access.** We partition memory into chapters and train a lightweight router that selects a small subset per input, adopting the conditional computation principle from Mixture of Experts (Shazeer et al., 2017; Fedus et al., 2021).
- **Scaling behavior under fixed compute.** We evaluate our memory architecture against standard transformers in iso-FLOP settings, complementing compute-optimal scaling discussions in language modeling (Hoffmann et al., 2022).

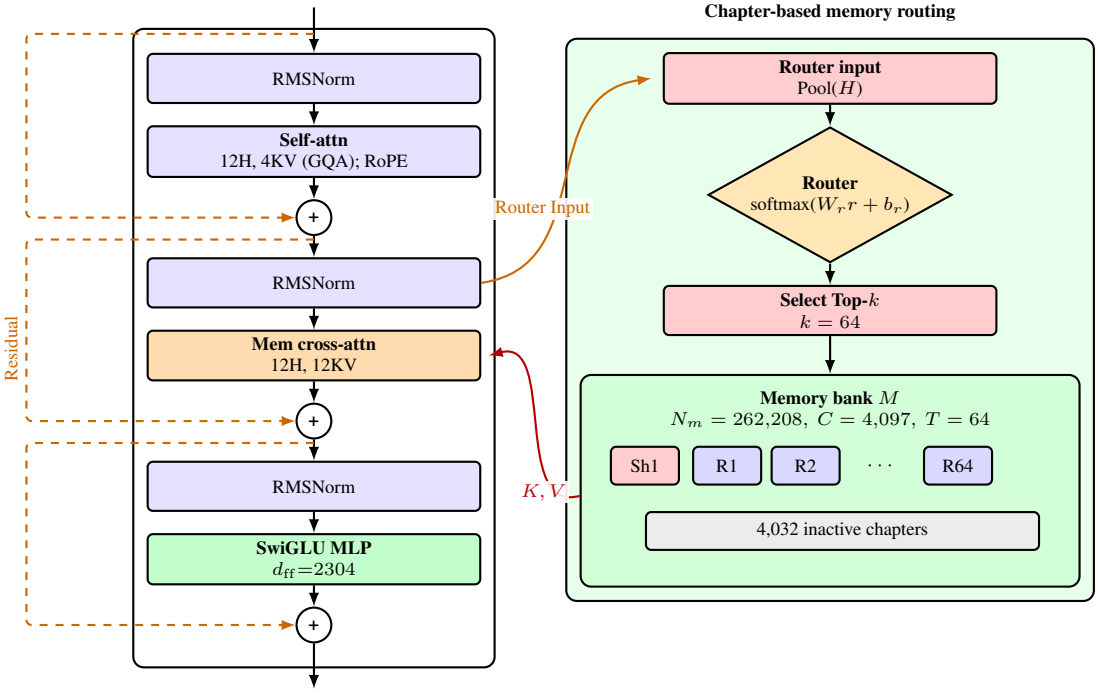


Figure 1: **Single transformer block with chapter-based memory routing**

- **Retention under continued training.** We provide evidence that explicit memory improves knowledge retention and reduces forgetting across training phase transitions, connecting to continual learning analyses in LLMs (Luo et al., 2023; Kirkpatrick et al., 2017).

5 Method

5.1 Memory layer: learned bank with chapter routing

We augment a standard decoder-only transformer (Vaswani et al., 2017) with a *memory layer* that provides persistent, addressable storage. The memory consists of a learned bank of latent tokens that are trained end-to-end, and accessed through cross-attention, following the general memory-augmented attention pattern (Sukhbaatar et al., 2019b; Wu et al., 2020).

Let $H \in \mathbb{R}^{B\times L\times d}$ be the token hidden states at a layer. We maintain a memory bank

$$M \in \mathbb{R}^{N_m \times d},$$

where each row is a learned memory token. The memory layer reads from M using cross-attention where the token stream produces queries and memory produces keys and values:

$$Q = \mathrm{Norm}(H)W_Q, \tag{1}$$
$$K = MW_K, \tag{2}$$
$$V = MW_V, \tag{3}$$
$$\mathrm{MemRead}(H, M) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_h}}\right)V. \tag{4}$$

The readout is added back to the token stream through a residual connection:

$$H' = H + \mathrm{MemRead}(H, M)$$

Table 1: Pretraining validation loss (lower is better).

Model	Val loss ↓
Vanilla (backbone-only)	2.92
Vanilla (iso-FLOP)	2.86
Mixture of Chapters (MoC)	**2.79**

5.2 Scaling the bank with chapters

Attending over all N_m memory tokens can be expensive for large banks. We therefore partition the bank into C *chapters*:

$$M = [M_1; \ldots; M_C], \quad M_c \in \mathbb{R}^{T \times d}, \quad N_m = CT.$$

For each input sequence, a lightweight router selects a small subset of chapters. We compute a sequence representation by pooling hidden states (for example, mean pooling):

$$r = \mathrm{Pool}(H) \in \mathbb{R}^d,$$

then score chapters and select the top-k:

$$p = \mathrm{softmax}(W_r r + b_r), \quad \mathcal{S} = \mathrm{TopK}(p, k).$$

The memory layer then attends only to the selected chapters:

$$M_{\mathcal{S}} = \mathrm{Concat}(\{M_c : c \in \mathcal{S}\}), \qquad H' = H + \mathrm{MemRead}(H, M_{\mathcal{S}}).$$

This reduces memory attention cost from $O(LN_m)$ to $O(LkT)$ per memory layer while preserving attention-based associative retrieval. Algorithm 1 (see Appendix A.3) summarizes the forward pass of a single MoC memory layer, including sequence-level routing, chapter selection, and routed memory cross-attention. The routing mechanism is inspired by sparse conditional computation in mixture-of-experts models (Shazeer et al., 2017; Fedus et al., 2022) and routing-based sparsification (Roy et al., 2021).

6 Experiments

6.1 Setup

Models. We compare (i) **Vanilla (iso-FLOP)**: a dense transformer baseline compute-matched to our memory model during pretraining (see Appendix A.2 for the analytic FLOPs calculation), (ii) **Mixture of Chapters (MoC)**: our memory-augmented transformer, and (iii) **Vanilla (backbone-only)**: the dense transformer backbone of the memory model with memory components removed.

Training protocol. We pretrain for 9,600 steps (9.6B tokens) and then instruction fine-tune (IFT) for 2 epochs on 230M tokens (3.2k steps). This is a relatively heavy post-training budget for this model scale and it induces clear forgetting in the dense baseline. During IFT, we increase context length from 1024 to 2048. All plots and reported pretraining results correspond to the 9,600-step run (not a continued run).

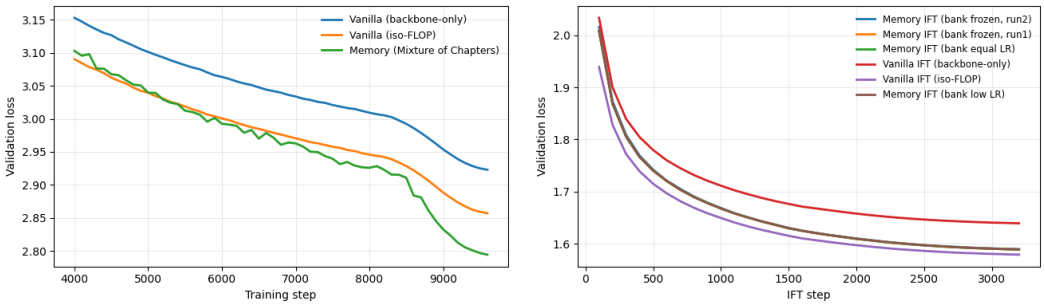
Evaluation. We report accuracy on MMLU (Hendrycks et al., 2021), ARC-Challenge (Clark et al., 2018), BoolQ (Clark et al., 2019), and OpenBookQA (Mihaylov et al., 2018). Random guessing is approximately 0.25 on ARC-Challenge and 0.50 on BoolQ.

6.2 Results

Pretraining loss. Table 1 shows validation loss at the end of pretraining. The memory model attains the best loss, and Figure 2 shows it continues to separate late in training, suggesting additional headroom with longer runs.

Table 2: Comparison of Vanilla, Mixture of Chapters (MoC) (Δ = IFT - pretrain; lower is better)

	Vanilla (iso-FLOP)			MoC		
Benchmark	Pretrain (%)	Δ (pp) $\downarrow$	IFT (%)	Pretrain (%)	Δ (pp) $\downarrow$	IFT (%)
MMLU	26.90	-0.99	25.91	**27.87**	**-0.35**	**27.52**
ARC-C	**31.77**	-6.69	25.08	31.44	**-2.68**	**28.76**
BoolQ	57.13	-6.24	50.89	**61.87**	**+0.24**	**62.11**
OBQA	36.20	-2.00	34.20	**37.40**	-2.00	**35.40**



(Left) Pretraining eval loss (9,600 steps). (Right) IFT eval loss for bank-freeze and LR variants.

Figure 2: Loss curves. Left: Pretraining. Right: IFT (Freezing the bank yields identical curves).

Benchmarks and forgetting under heavy IFT. Table 2 reports benchmark accuracy after pretraining and after IFT, along with deltas (IFT minus pretrain). Vanilla is strongly affected on knowledge-heavy tasks: ARC-Challenge drops by 0.0669 (0.3177 $\rightarrow$ 0.2508, approaching random) and BoolQ drops by 0.0624 (0.5713 $\rightarrow$ 0.5089, approaching random). In contrast, the memory model remains stable on these tasks: BoolQ is effectively unchanged (+0.0024), and ARC-Challenge drops only 0.0268, less than half the vanilla degradation. This pattern is consistent with the memory bank anchoring factual knowledge during continued training.

IFT loss and freezing the memory bank. Figure 2 (right) compares IFT validation loss for memory variants where the bank is frozen, trained with a very small learning rate, or trained with the same learning rate as the backbone. The curves overlap closely, and post-IFT benchmark scores remain within noise across these settings (Appendix A.4). This indicates the pretrained memory bank can be reused during post-training without further bank updates.

Pretraining distribution after IFT. Evaluating IFT checkpoints on the pretraining validation distribution yields higher loss for the memory model (3.2024) than vanilla (3.0574), despite better retention on knowledge benchmarks. This suggests the memory model can shift further toward instruction alignment while preserving factual knowledge, consistent with a separation of roles between backbone adaptation and memory anchoring.

7 Conclusion

We introduced **Mixture of Chapters (MoC)**, a memory-augmented transformer that adds a learned latent memory bank and scales access via sequence-level chapter routing, providing an explicit associative memory substrate with sparse, tractable access. In a 9.6B-token pretraining run, the memory model achieves lower validation loss than compute-matched dense baselines and improves knowledge-heavy benchmark performance. Under heavy post-training (2-epoch IFT on 230M tokens with context length increased from 1024 to 2048), the dense baseline exhibits substantial forgetting on ARC-Challenge and BoolQ, while the memory model remains comparatively stable; freezing the bank during IFT performs on par with updating it. Overall, these results point to learned memory as a complementary axis of scaling and motivate further study of larger banks, longer training, and improved routing/organization for specialization and interpretability.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, I. Barr, Y. Hasson, et al. Flamingo: A visual language model for few-shot learning. arXiv preprint arXiv:2204.14198, 2022.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time, 2025. URL https://arxiv.org/abs/2501.00663.

Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. *arXiv preprint arXiv:2412.09764*, 2024.

Vincent-Pierre Berges, Barlas Oguz, Daniel Haziza, Wen-Tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://proceedings.mlr.press/v267/berges25a.html.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, et al. Improving language models by retrieving from trillions of tokens. arXiv preprint arXiv:2112.04426, 2021.

Alexander Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Recurrent memory transformer. arXiv preprint arXiv:2207.06881, 2022.

Alexander Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Scaling transformer to 1M tokens and beyond with RMT. arXiv preprint arXiv:2304.11062, 2023.

Lingfeng Chen, Zhenyu Zhang, Ying Sheng, Lin Dong, Weijia Xu, Lianmin Zheng, Bohan Zhuang, Yukang Cao, Zhangyang Wang, and Beidi Chen. NaCl: A general and effective KV cache eviction framework for LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://aclanthology.org/2024.acl-long.428/.

Xin Cheng et al. Engram: Conditional memory via scalable lookup: A new axis of sparsity for large language models, 2026. URL https://arxiv.org/abs/2601.07372.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL-HLT*, 2019. URL https://arxiv.org/abs/1905.10044.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457.

Andrea Cossu, Tinne Tuytelaars, Antonio Carta, Lucia Passaro, Vincenzo Lomonaco, and Davide Bacciu. Continual pre-training mitigates forgetting in language and vision. *arXiv preprint arXiv:2205.09357*, 2022.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019a. URL https://aclanthology.org/P19-1285/.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. arXiv preprint arXiv:1901.02860, 2019b.

Michiel de Jong, Yury Zemlyanskiy, Nicholas FitzGerald, Fei Sha, and William Cohen. Mention memory: Incorporating textual knowledge into transformers through entity mention attention, 2021a. URL https://arxiv.org/abs/2110.06176.

Michiel de Jong, Yury Zemlyanskiy, Nicholas FitzGerald, Fei Sha, and William Cohen. Mention memory: Incorporating textual knowledge into transformers through entity mention attention. *arXiv preprint arXiv:2110.06176*, 2021b.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL https://jmlr.org/papers/v23/21-0998.html.

Thibault Fevry, Livio Baldini Soares, Nicholas FitzGerald, Eunsol Choi, and Tom Kwiatkowski. Entities as experts: Sparse memory access with entity supervision. arXiv preprint arXiv:2004.07202, 2020.

N. Fountas, Nicola De Cao, T. Wolff, and Maxime Peyrard. Human-inspired episodic memory for infinite context LLMs, 2025. URL https://arxiv.org/abs/2407.09450.

T. Ge, J. Hu, L. Li, X. Qiu, X. Huang, and L. Si. In-context autoencoder for context compression in a large language model. arXiv preprint arXiv:2307.06945, 2023.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2009.03300.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew Botvinick, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver io: A general architecture for structured inputs and outputs. *arXiv preprint arXiv:2107.14795*, 2021a. URL https://arxiv.org/abs/2107.14795.

Andrew Jaegle, Felix Gimeno, Andy Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention. arXiv preprint arXiv:2103.03206, 2021b.

Junhao Kang, Wei Wu, Filippos Christianos, Arthur J. Chan, Frank Greenlee, George Thomas, Mahdi Purtorab, and Adam Toulis. LM2: Large memory models, 2025. URL https://arxiv.org/abs/2502.06049.

Mahdi Karami, Ali Behrouz, Praneeth Kacham, and Vahab Mirrokni. Trellis: Learning to compress key-value memory in attention models. *arXiv preprint arXiv:2512.23852*, 2025. URL https://arxiv.org/abs/2512.23852. In COLM 2025.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*, 2019. URL https://arxiv.org/abs/1911.00172.

J. H. Kim, J. Ma, L. Lausen, C. Tan, J. Zhang, and Y. Cui. Compressed context memory for online language model interaction. arXiv preprint arXiv:2312.03414, 2024.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. In *Proceedings of the National Academy of Sciences*, 2017. URL https://www.pnas.org/doi/10.1073/pnas.1611835114.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. *arXiv preprint arXiv:1907.05242*, 2019a.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems*, 2019b. URL https://arxiv.org/abs/1907.05242.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *arXiv preprint arXiv:2005.11401*, 2020a.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, 2020b. URL https://arxiv.org/abs/2005.11401.

Peng Liu, Y. Zhang, and Daniel S. Weld. Larimar: Large language models with episodic memory control, 2024. URL https://arxiv.org/abs/2403.11901.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*, 2023.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018. URL https://arxiv.org/abs/1809.02789.

Tsendsuren Munkhdalai, Manaal Faruqui, and Saumya Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. arXiv preprint arXiv:2404.07143, 2024.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. arXiv preprint arXiv:1911.05507, 2020.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *arXiv preprint arXiv:2003.05997*, 2020.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021. URL https://aclanthology.org/2021.tacl-1.4/.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023. URL https://arxiv.org/abs/2302.04761.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017. URL https://arxiv.org/abs/1701.06538.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Augmenting self-attention with persistent memory. arXiv preprint arXiv:1907.01470, 2019a.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Hervé Jégou, and Armand Joulin. Augmenting self-attention with persistent memory. *arXiv preprint arXiv:1907.01470*, 2019b. URL https://arxiv.org/abs/1907.01470.

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

Patrick Verga, Haitian Sun, Livio Baldini Soares, and William Cohen. Facts as experts: Adaptable and interpretable neural memory over symbolic knowledge. arXiv preprint arXiv:2007.00849, 2020.

Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. MEMORYLLM: Towards self-updatable large language models, 2024. URL https://arxiv.org/abs/2402.04624.

Yu Wang et al. M+: Extending MEMORYLLM with scalable long-term memory, 2025. URL https://arxiv.org/abs/2502.00592.

Qingyang Wu and Zhou Yu. Stateful memory-augmented transformers for efficient dialogue modeling. arXiv preprint arXiv:2209.07634, 2022.

Y. Wu, Y. Zhao, B. Hu, P. Minervini, P. Stenetorp, and S. Riedel. An efficient memory-augmented transformer for knowledge-intensive NLP tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022a. arXiv:2210.16773.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memformer: A memory-augmented transformer for sequence modeling. *arXiv preprint arXiv:2010.06891*, 2020. URL https://arxiv.org/abs/2010.06891.

Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022b.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.17453.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

P. Zhang, H. Qian, Q. Ye, and Z. Dou. Long context compression with activation beacon. arXiv preprint arXiv:2401.03462, 2024.

Z. Zhang, W. Shao, Y. Ge, X. Wang, J. Gu, and P. Luo. Cached transformers: Improving transformers with differentiable memory cache. arXiv preprint arXiv:2312.12742, 2023a.

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. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023b. URL https://arxiv.org/abs/2306.14048.

A Appendix

A.1 Extended literature review (concise map)

Due to the page limit, the main paper focuses on the closest comparisons (Titans, Memory Layers at Scale, PKM). This appendix provides a compact narrative overview of broader related directions, complemented by Table 3.

Inference-time and external memory. A common strategy is to externalize memory at inference time, either by caching past activations to improve efficiency and extend effective context, or by retrieving text from an external index. Transformer-XL is a canonical example of recurrence and caching (Dai et al., 2019b), while RAG exemplifies retrieval-conditioned generation (Lewis et al., 2020a). These approaches are strong baselines, but the stored content remains outside the model and must be fetched and integrated per query.

Learned memory inside the model. A complementary line integrates memory as a trainable component. Memformer reads and writes through an internal memory mechanism coupled to token representations (Wu et al., 2020). For scaling, PKM provides efficient large key-value lookup (Lample et al., 2019a), and Memory Layers at Scale shows that trainable memory layers can add substantial capacity with near-constant FLOPs and strong factual gains (Berges et al., 2025). Our work is closest to this direction, but uses a learned latent-token bank accessed by cross-attention and scales access via sequence-level chapter routing.

Updatable and test-time memory. Some methods emphasize explicit test-time updates or state that evolves across interactions, such as Titans (Behrouz et al., 2025). These approaches often target persistent personalization or long-horizon interaction, whereas our focus is on scaling learned associative memory trained end-to-end and analyzing retention under continued training.

Sparse routing and conditional computation. Routing and conditional computation provide a general template for scaling by activating only a subset of components. MoE systems route inputs to a small subset of experts (Shazeer et al., 2017; Fedus et al., 2022). Our method borrows this principle but applies routing to memory selection, with chapters serving as routed memory units.

Retention and catastrophic forgetting. Catastrophic forgetting is a classical issue in sequential training (Kirkpatrick et al., 2017), and has been measured in continual or sequential tuning of language models. Our experiments connect this literature to architectural memory by testing whether a learned bank helps preserve factual knowledge across the pretraining-to-IFT transition.

Table 3: Literature map (representative, not exhaustive).

Category	Representative works
Learnable internal memory bank (cross-attn / persistent tokens)	Memformer (Wu et al., 2020), LM2 (Kang et al., 2025), persistent memory vectors (Sukhbaatar et al., 2019a), stateful memory variants (Wu & Yu, 2022).
Large trainable memory layers (scalable access)	Memory Layers at Scale (Berges et al., 2025), PKM (Lample et al., 2019a), EMAT (Wu et al., 2022a), Memorizing Transformers (Wu et al., 2022b).
Test-time / updatable memory	Titans (Behrouz et al., 2025), MemoryLLM (Wang et al., 2024), M+ (Wang et al., 2025), Larimar (Liu et al., 2024), EM-LLM (Fountas et al., 2025), Engram (Cheng et al., 2026).
Recurrence and compressive memory for long context	Transformer-XL (Dai et al., 2019b), RMT (Bulatov et al., 2022; 2023), Compressive Transformers (Rae et al., 2020), Infini-attention (Munkhdalai et al., 2024), Cached Transformers (Zhang et al., 2023a).
Structured/entity memories	Mention Memory (de Jong et al., 2021a), Entities as Experts (Fevry et al., 2020), Facts as Experts (Verga et al., 2020).
Latent-token architectures and context compression	Perceiver/Perceiver IO (Jaegle et al., 2021b;a), Flamingo (Alayrac et al., 2022), Activation Beacon (Zhang et al., 2024), ICAE (Ge et al., 2023), compressed-context memory (Kim et al., 2024).
Retrieval and KV-cache baselines	RAG (Lewis et al., 2020a), RETRO (Borgeaud et al., 2021), kNN-LM (Khandelwal et al., 2019), KV-cache retention/compression (H_2O, StreamingLLM, NACL, Trellis) (Zhang et al., 2023b; Xiao et al., 2024; Chen et al., 2024; Karami et al., 2025).

A.2 Analytic FLOPs calculation

In this section, we estimate FLOPs analytically for a *single sequence* with batch size $B = 1$ and sequence length $L = 1024$. Our codebase includes a reference implementation, `estimate_flops.py`, which follows the same counting rules used here for complete training runs. The derivation below mirrors that implementation, while making the layer-by-layer calculation explicit.

Linear layers use

$$\mathrm{FLOPs}_{\mathrm{linear}} = 2\,N\,d_{\mathrm{in}}\,d_{\mathrm{out}},$$

attention matmuls use

$$\mathrm{FLOPs}_{\mathrm{attn\ matmul}} = 4\,B\,L_q\,L_k\,d,$$

RMSNorm uses

$$\mathrm{FLOPs}_{\mathrm{RMSNorm}} = N(4d + 4),$$

and the backward pass is approximated as $2\times$ the forward pass, consistent with the estimator.

For our backbone, $d = 768$, $d_{\mathrm{ff}} = 2304$, $h = 12$, $h_{\mathrm{kv}} = 4$, so the self-attention KV dimension is

$$d_{\mathrm{kv}} = h_{\mathrm{kv}} \cdot \frac{d}{h} = 4 \cdot 64 = 256.$$

Standard transformer layer. A standard layer contains self-attention, two RMSNorms, one SwiGLU MLP, RoPE, and two residual additions.

Self-attention projections and attention.

$$Q : 2 \cdot 1024 \cdot 768 \cdot 768 = 1{,}207{,}959{,}552 \tag{5}$$
$$K : 2 \cdot 1024 \cdot 768 \cdot 256 = 402{,}653{,}184 \tag{6}$$
$$V : 2 \cdot 1024 \cdot 768 \cdot 256 = 402{,}653{,}184 \tag{7}$$
$$O : 2 \cdot 1024 \cdot 768 \cdot 768 = 1{,}207{,}959{,}552 \tag{8}$$
$$QK^\top,\ AV : 4 \cdot 1 \cdot 1024 \cdot 1024 \cdot 768 = 3{,}221{,}225{,}472. \tag{9}$$

The estimator also includes attention softmax, masking, and scaling:

$$12 \cdot 1024 \cdot 1024 \cdot (5 + 2) = 88{,}080{,}384.$$

Hence total self-attention FLOPs per standard layer are

$$6{,}530{,}531{,}328.$$

RoPE.

$$3 \cdot 1024 \cdot (768 + 256) = 3{,}145{,}728.$$

Two RMSNorms.

$$2 \cdot 1024 \cdot (4 \cdot 768 + 4) = 6{,}299{,}648.$$

SwiGLU MLP.

$$W_{\text{up}} : 2 \cdot 1024 \cdot 768 \cdot 2304 = 3{,}623{,}878{,}656 \tag{10}$$
$$W_{\text{gate}} : 2 \cdot 1024 \cdot 768 \cdot 2304 = 3{,}623{,}878{,}656 \tag{11}$$
$$W_{\text{down}} : 2 \cdot 1024 \cdot 2304 \cdot 768 = 3{,}623{,}878{,}656 \tag{12}$$
$$\text{SwiGLU activation} : 1024 \cdot 2304 \cdot 5 = 11{,}796{,}480. \tag{13}$$

So total MLP FLOPs per layer are

$$10{,}883{,}432{,}448.$$

Residual adds.

$$2 \cdot 1024 \cdot 768 = 1{,}572{,}864.$$

Putting these together, one *standard* transformer layer costs

$$6{,}530{,}531{,}328 + 3{,}145{,}728 + 6{,}299{,}648 + 10{,}883{,}432{,}448 + 1{,}572{,}864 = 17{,}424{,}982{,}016$$

FLOPs.

MoC memory layer. An MoC memory layer adds router computation, memory preprocessing, memory cross-attention, an extra RMSNorm, and an extra residual add.

The memory bank has $262{,}208$ tokens, partitioned into 4097 chapters, giving

$$T = 262{,}208/4097 = 64$$

tokens per chapter. With one shared chapter and top-$k = 64$ routed chapters, the selected memory tokens per sequence are

$$N_{\text{sel}} = (1 + 64) \cdot 64 = 4160.$$

Router forward. For sequence-level routing, the estimator includes sequence pooling, router projection, softmax, and top-k:

$$\text{pooling} : 1 \cdot 768 \cdot (1024 - 1) + 768 = 786{,}432 \tag{14}$$
$$\text{router linear} : 2 \cdot 1 \cdot 768 \cdot 4097 = 6{,}292{,}992 \tag{15}$$
$$\text{router softmax} : 4097 \cdot 5 = 20{,}485 \tag{16}$$
$$\text{top-}k : 4097 \cdot \lceil \log_2 64 \rceil = 24{,}582. \tag{17}$$

Thus router forward FLOPs are

$$7{,}124{,}491.$$

Router auxiliary-loss compute. The estimator also includes the auxiliary routing-loss terms (load-balancing, z-loss, entropy, and related reductions), which contribute

$$331{,}859$$

FLOPs per memory layer.

Memory preprocessing. The estimator includes chapter weighting plus normalization of the selected memory tokens:

$$\text{weighting} : 1 \cdot 4160 \cdot 768 = 3{,}194{,}880 \tag{18}$$
$$\text{RMSNorm} : 4160 \cdot (4 \cdot 768 + 4) = 12{,}796{,}160. \tag{19}$$

So memory preprocessing contributes

$$15{,}991{,}040$$

FLOPs.

Memory attention. Here $h_{\text{mem}} = 12$, $h_{\text{mem,kv}} = 12$, so memory KV dimension is 768.

$$Q : 2 \cdot 1024 \cdot 768 \cdot 768 = 1{,}207{,}959{,}552 \tag{20}$$
$$K : 2 \cdot 4160 \cdot 768 \cdot 768 = 4{,}907{,}335{,}680 \tag{21}$$
$$V : 2 \cdot 4160 \cdot 768 \cdot 768 = 4{,}907{,}335{,}680 \tag{22}$$
$$O : 2 \cdot 1024 \cdot 768 \cdot 768 = 1{,}207{,}959{,}552 \tag{23}$$
$$QK^\top,\ AV : 4 \cdot 1 \cdot 1024 \cdot 4160 \cdot 768 = 13{,}086{,}351{,}360. \tag{24}$$

The attention softmax, masking, and scaling term is

$$12 \cdot 1024 \cdot 4160 \cdot (5 + 2) = 357{,}826{,}560.$$

Hence memory attention contributes

$$25{,}674{,}645{,}504$$

FLOPs.

Extra norm and residual.

$$\text{extra RMSNorm} = 1024 \cdot (4 \cdot 768 + 4) = 3{,}149{,}824, \qquad \text{extra residual} = 1024 \cdot 768 = 786{,}432.$$

Therefore the *extra* cost added by an MoC memory layer is

$$7{,}124{,}491 + 331{,}859 + 15{,}991{,}040 + 25{,}674{,}645{,}504 + 3{,}149{,}824 + 786{,}432 = 25{,}702{,}029{,}150.$$

Adding the underlying standard transformer layer gives

$$17{,}424{,}982{,}016 + 25{,}702{,}029{,}150 = 43{,}127{,}011{,}166$$

FLOPs per MoC memory layer.

Final prediction head and loss. After the transformer stack, the estimator includes a final RMSNorm, LM head, and cross-entropy loss:

$$\text{final RMSNorm} : 1024 \cdot (4 \cdot 768 + 4) = 3{,}149{,}824 \tag{25}$$
$$\text{LM head} : 2 \cdot 1024 \cdot 768 \cdot 49152 = 77{,}309{,}411{,}328 \tag{26}$$
$$\text{CE loss} : (1024 - 1) \cdot 49152 \cdot 5 = 251{,}412{,}480. \tag{27}$$

Thus the final prediction/loss block contributes

$$77{,}563{,}973{,}632$$

FLOPs.

Table 4: Per-sequence FLOPs for one forward pass and one forward+backward pass ($B = 1, L = 1024$).

Model	Forward FLOPs	Backward FLOPs	Forward+Backward FLOPs
Vanilla (backbone-only)	0.356T	0.713T	1.069T
Vanilla (iso-FLOP)	0.496T	0.992T	1.487T
Mixture of Chapters (MoC)	0.459T	0.918T	1.378T

Model totals. The backbone-only vanilla model has 16 standard layers:

$$F_{\text{fwd}}^{\text{backbone}} = 16 \cdot 17{,}424{,}982{,}016 + 77{,}563{,}973{,}632 = 356{,}363{,}685{,}888.$$

The iso-FLOP vanilla baseline has 24 standard layers:

$$F_{\text{fwd}}^{\text{iso}} = 24 \cdot 17{,}424{,}982{,}016 + 77{,}563{,}973{,}632 = 495{,}763{,}542{,}016.$$

The memory model has 12 standard layers and 4 MoC memory layers:

$$F_{\text{fwd}}^{\text{MoC}} = 12 \cdot 17{,}424{,}982{,}016 + 4 \cdot 43{,}127{,}011{,}166 + 77{,}563{,}973{,}632 = 459{,}171{,}802{,}488.$$

Following the estimator, we approximate one backward pass as $2\times$ the forward FLOPs:

$$F_{\text{bwd}} \approx 2F_{\text{fwd}}.$$

Hence the total for one forward plus one backward pass is $3F_{\text{fwd}}$.

Note that these are *single-sequence* estimates. They differ slightly from dividing the microstep estimates by batch size, because the sequence-level router auxiliary-loss terms do not scale exactly linearly with batch size.

A.3 Algorithm: Mixture of Chapters (MoC) memory layer

Algorithm 1 MoC memory layer with sequence-level routing

Require: Batch hidden states $H \in \mathbb{R}^{B\times L\times d}$
Require: Chaptered memory bank $M = [M_1; \ldots; M_C]$, where $M_c \in \mathbb{R}^{T\times d}$
Require: Router parameters W_r, b_r, projections W_Q, W_K, W_V, top-k
Ensure: Updated hidden states $H' \in \mathbb{R}^{B\times L\times d}$

for $b = 1$ to B **do**
  $r_b \leftarrow \text{Pool}(H_b)$
  $p_b \leftarrow \text{softmax}(W_r r_b + b_r)$
  $\mathcal{S}_b \leftarrow \text{TopK}(p_b, k)$
  $M_b \leftarrow \text{Concat}(\{M_c : c \in \mathcal{S}_b\})$
  $Q_b \leftarrow \text{Norm}(H_b)W_Q$
  $K_b \leftarrow M_b W_K$
  $V_b \leftarrow M_b W_V$
  $A_b \leftarrow \text{softmax}\Big(\frac{Q_b K_b^\top}{\sqrt{d_h}}\Big)$
  $\text{MemRead}_b \leftarrow A_b V_b$
  $H'_b \leftarrow H_b + \text{MemRead}_b$
end for
return $H' = [H'_1; \ldots; H'_B]$

A.4 Additional experimental details

A.4.1 Model sizes

The memory model has a larger total parameter count due to the persistent bank, but uses sparse chapter routing so only a subset of the bank is activated per input. We therefore match pretraining compute to a dense baseline (Vanilla iso-FLOP). For the non-memory baselines, the **Vanilla**

Table 5: Main MoC architecture used in our experiments.

Component	Setting
Backbone	Decoder-only transformer with 16 layers, hidden size 768, 12 attention heads, 4 KV heads, SwiGLU MLP with intermediate size 2304, RMSNorm, RoPE with $\theta = 100000$, tied embeddings, vocabulary size 49152
Memory placement	Memory cross-attention inserted at layers $\{2, 6, 10, 14\}$
Memory bank	262,208 latent memory tokens, shared across memory layers
Chapter routing	4097 total chapters, including 1 shared chapter; top-$k = 64$ chapters selected per sequence
Memory attention	12 memory query heads and 12 memory KV heads
Routing details	Sequence-level routing, routed scaling factor 2.5, shared/routed mixing normalized with RMSNorm
Regularization	Load-balance loss coefficient 0.01, z-loss coefficient 0.001
Other	Memory adapter enabled, memory initialization std. 0.02, memory quantization disabled

Table 6: Main training settings for pretraining and IFT.

Setting	Pretraining	IFT
Tokenizer	`HuggingFaceTB/SmolLM2-135M`	`HuggingFaceTB/SmolLM2-135M-Instruct`
Dataset	`Tasmay-Tib/fineweb-edu-10bt-split`	`HuggingFaceH4/ultrachat_200k`
Data split	train / eval	train_sft / test_sft
Sequence length	1024	2048
Distributed setup	DDP on 8 GPUs	DDP on 4 GPUs
Batch size	32 with gradient accumulation 4	32 with gradient accumulation 1
Training duration	9600 steps	2 epochs (3200 steps)
Optimizer	AdamW	AdamW
Learning rates	base model: 3×10^{-4} memory layers: 6×10^{-4} memory bank: 6×10^{-4}	base model: 3×10^{-5} memory layers: 1.5×10^{-5} memory bank: 5×10^{-6}
Scheduler	WSD, warmup 250 steps, min LR ratio 0.1, decay start at step 8160	Cosine, warmup 250 steps
Optimization details	weight decay 0.1, Adam $\beta_1 = 0.9$, $\beta_2 = 0.95$, grad clip 1.0	weight decay 0.1, Adam $\beta_1 = 0.9$, $\beta_2 = 0.95$, grad clip 1.0
Precision	bf16	bf16
Checkpointing	gradient checkpointing enabled	gradient checkpointing enabled
Initialization	random initialization	initialized from the pretrained MoC checkpoint

(backbone-only) model uses the same 16-layer backbone architecture with the memory layers removed, while the **Vanilla (iso-FLOP)** baseline is a denser variant obtained by repeating the standard backbone layers without memory modules and increasing depth to match the pretraining compute budget of the memory model. For reference, the dense baseline has 202.94M parameters, while the memory model has 371.29M total parameters composed of a 147.87M backbone plus a 201.38M bank and 22.04M memory-layer parameters.

A.4.2 Model and training configuration

Tables 5 and 6 summarize the main Mixture of Chapters (MoC) architecture and the training settings used for pretraining and instruction fine-tuning (IFT).

For comparison, the vanilla backbone-only baseline uses the same 16-layer backbone with memory disabled, while the iso-FLOP vanilla baseline increases depth to 24 layers.

Table 7: Benchmark accuracy after pretraining (including backbone-only).

Model	MMLU	ARC-C	BoolQ	OBQA
Vanilla (iso-FLOP)	0.2690	0.3177	0.5713	0.3620
MoC	**0.2787**	0.3144	**0.6187**	**0.3740**
Vanilla (backbone-only)	0.2746	**0.3311**	0.5446	0.3340

Table 8: MoC IFT benchmarks for bank update settings.

IFT setting	MMLU	ARC-C	BoolQ	OBQA
MoC (bank frozen)	**0.2754**	0.2843	**0.6214**	**0.3540**
MoC (bank 1/10 LR)*	0.2752	0.2876	0.6211	**0.3540**
MoC (bank equal LR)	0.2753	**0.2943**	0.6211	0.3480

* *denotes the main setting reported above.*

A.4.3 Pretraining benchmarks including the backbone-only baseline

Table 7 includes the backbone-only benchmark results to contextualize gains from MoC versus simply changing backbone capacity.

A.4.4 IFT learning rates and bank update ablations

IFT uses a base learning rate of 3×10^{-5} (one tenth of the pretraining rate). We vary the memory bank learning rate across: (1) **Frozen bank** (LR = 0), (2) **Low bank LR** (very small LR), (3) **Equal LR** (bank LR matches base LR). Across these settings, Figure 2 shows no visible divergence in validation loss curves.

Table 8 reports benchmark scores for representative settings. Freezing the bank performs on par with updating it, indicating that the pretrained bank is retained and sufficient for post-training.

A.4.5 IFT shifts the pretraining distribution loss while preserving knowledge benchmarks

When evaluating IFT checkpoints on the pretraining validation distribution, the memory model exhibits higher loss (3.2024) than the vanilla baseline (3.0574), even though it retains benchmark performance substantially better on ARC-Challenge and BoolQ. This suggests that the memory model can adapt more strongly toward the IFT objective while keeping factual knowledge stable through the explicit bank, consistent with a division of labor between backbone adaptation and memory anchoring.

A.4.6 Compute and memory implications during post-training

Freezing the memory bank during IFT removes bank updates and reduces optimizer state and gradient memory for that component, which lowers VRAM requirements. In addition, increasing context length from 1024 to 2048 increases self-attention cost for all models. Under this longer-context setting, the benefit of freezing the bank is preserved while the dense baseline pays the full quadratic self-attention cost. This combination makes the memory approach attractive for heavier or longer post-training schedules.

A.5 Ablation studies and future work

This section lists ablations for *Mixture of Chapters* and outlines directions to clarify scaling, efficiency, and retention.

Ablation studies.

- **Top-k routing sweep.** Vary the number of routed chapters, e.g., $k \in \{1, 2, 4, 8, 16, 32, 64\}$, holding bank size fixed to map the quality–compute frontier.
- **Routing granularity and train/inference mismatch.** Compare (i) sequence routing at train and inference (current), (ii) token routing at inference only, and (iii) token routing at train and inference (if feasible).
- **Shared chapters.** Sweep the number of always-on shared chapters and mixing schemes: $n_{\text{shared}} \in \{0, 1, 2, 4, 8\}$.
- **Memory placement and sharing.** Study where memory layers are inserted and whether the bank is shared across layers vs per-layer banks.
- **Router regularization.** Sweep load-balancing / auxiliary routing losses (and any z-loss) to quantify utilization collapse vs quality tradeoffs.
- **Initialization.** Ablate output-projection initialization (e.g., zero-init) and bank initialization scale to test stability and early training behavior.
- **Memory size and chapter geometry.** Scale total memory tokens, number of chapters, and tokens-per-chapter T to separate "more chapters" from "longer chapters."
- **Compression ablations.** Evaluate quantized banks and low-rank factorization of bank/projections to trade expressiveness for capacity.
- **PEFT comparisons: LoRA vs memory adapters vs both.** Compare LoRA-only, memory-adapter-only, and combined settings under matched trainable parameter budgets.

Future work.

- **Scaling laws for learned memory.** Extend pretraining beyond 9.6B tokens and sweep bank capacity to quantify how performance scales with memory under fixed compute.
- **Token-level routing with efficient kernels.** Investigate kernels that avoid materializing large routed key tensors, enabling token routing during training without large memory overhead.
- **Dynamic and test-time updates.** Add controlled write/update mechanisms to support continual acquisition and personalization while retaining the "freeze bank" benefits.
- **Interpretability and chapter-level editing.** Analyze router selections for specialization, and test targeted chapter edits (zeroing, swapping, pruning) to localize knowledge.
- **Hybrid memory: shared global + routed private.** Explore explicit separation between a small shared global memory and a large routed bank, with different update schedules or learning rates.
- **Broader evaluations.** Add tests for long-context generalization, factual recall, and retention under multiple sequential post-training stages.
- **Interpretability and analysis of memory usage.** Analyze how memory is used across layers and time: (i) chapter selection statistics (entropy, sparsity, stability across prompts), (ii) per-layer reliance on memory (attention mass to memory vs self-attn), (iii) chapter specialization (clustering chapters by the queries they serve or by input domains), and (iv) causal interventions such as zeroing/swapping/pruning selected chapters or memory tokens to localize which knowledge is stored where. This can be paired with qualitative retrieval probes by decoding nearest-neighbor token projections from memory to inspect what each chapter appears to represent.