Training Transformers for KV Cache Compressibility

Anonymous Author(s)
Affiliation
Address
email

Abstract

Long-context language modeling is increasingly constrained by the Key–Value (KV) cache, whose memory and decode-time access costs scale linearly with the prefix length. This bottleneck has motivated a range of context-compression methods, from token-level summarization to recent optimization-based KV cache compression methods. These post-hoc methods operate on the KV cache of a fixed pretrained model, so their effectiveness is fundamentally limited by how well the model's internal representations can be compressed. In this work, we formalize the notion of KV compressibility and show that it is a property of the *learned representations*, rather than of the context alone. We prove that almost any sequence-to-vector function admits both highly compressible and inherently non-compressible transformer implementations, highlighting the need to guide transformers toward compressible representations during training. Motivated by this, we propose **KV**-**C**ompression **A**ware **T**raining (KV-CAT), a continued pretraining procedure that incentivizes the emergence of compressible representations. We introduce a train-time KV sparsification policy that masks KV slots during training. This forces the model to use fewer KV slots and encourages it to learn representations amenable to post-hoc compression. Empirically, we show that KV-CAT improves the quality–budget tradeoff of downstream compression methods across retrieval, long-context question answering, and perplexity-based evaluation of compressed-prefix continuation.

1 Introduction

Language models (LMs) are increasingly deployed in long-horizon settings, from understanding large codebases, long-form documents, and personal data repositories [41; 23; 1], to long-form reasoning and continuously learning agents [8; 60; 35]. Using autoregressive transformer-based LMs in these settings introduces a significant memory bottleneck: the Key–Value (KV) cache. During inference, these models must store the key and value vectors for every token, at every layer and (KV-)attention head. For long sequences, this cache can dominate both memory usage and decoding cost, turning context length into a primary serving bottleneck [59; 34; 52].

A substantial body of recent work has sought to mitigate this bottleneck, broadly falling into two categories. The first category focuses on designing more efficient alternatives to transformers. Examples include linear attention mechanisms [29; 11; 20; 54], state space models [18; 17], sparse attention variants [3; 57], and more. These methods significantly reduce the computational cost of long-context language modeling. However, this efficiency typically comes at the cost of empirical performance, and such models still lag behind transformers at scale.

The second category focuses on inference-time interventions applied to a fixed pretrained transformer. These methods either operate on the input context or, more generally, directly on the KV cache. Early approaches are largely heuristic, including textual summarization [46; 42], learned token filtering [24], and policies based on attention patterns, recency, heavy-hitter behavior, or layer-wise

importance [59; 52; 34; 6]. More recently, optimization-based KV cache compression[1] methods have emerged as powerful techniques. For example, Eyuboglu et al. [16] use gradient-based optimization to match the distribution induced by the original cache, while Zweiger et al. [61] employ layer-wise objectives to reproduce its attention traces.

The success of KV cache compression suggests that the full KV cache often contains redundancies. However, as most approaches operate on a **fixed model**, they are inherently limited by how well that particular model's representations can be compressed. Crucially, this compressibility is not determined by the input sequence alone: two transformers can use very different internal representations when processing the same sequence, yet have identical next-token distributions. Consequently, some transformers' KV caches may be more amenable to compression than others.

This work. In this paper, we study the following question: can transformer LMs be trained in a way that leads to KV caches that are more amenable to post-hoc compression? This shifts the target from the KV cache compression algorithm to the model itself. To address this question, we introduce the notion of **KV-compressibility**. Informally, a transformer is KV-compressible if there exists a compression policy that maps the KV cache of long input sequences to a shorter KV cache while preserving the model's next token distribution.

Our theoretical results show that for almost any sequence-to-vector function, there exist transformer implementations whose prefix can be compressed to a single KV pair, as well as implementations for which any non-trivial compression incurs a constant error. To build intuition, we consider a motivating example: character histogram computation. We show that natural transformer implementations of this computation may produce incompressible token representations, whereas more structured alternatives are highly compressible. This motivates treating KV compressibility as an explicit training objective.

Guided by this perspective, we propose **KV**-**C**ompression **A**ware **T**raining (KV-CAT), a continued pretraining (CPT) procedure that promotes the emergence of KV-compressible internal representations. Starting from a pretrained transformer, KV-CAT introduces a train-time KV sparsification policy which masks out a constant fraction of the KV slots. The training objective combines a self-distillation loss, which matches the masked model's distribution to the dense model's distribution, and an NTP loss applied to the unmasked forward pass to preserve uncompressed model behavior. This exposes the model to the information bottleneck induced by KV cache compression during training, encouraging it to reorganize its representations into a more compressible form while maintaining performance in the uncompressed setting. Importantly, rather than replacing post-hoc compression methods, KV-CAT makes transformers more amenable to them.

To evaluate KV-CAT, we apply it to QWEN2.5 models [51] via continuous pretraining and test state-of-the-art optimization-based KV compression methods on the resulting checkpoints. We evaluate suffix completion under compressed prefixes, retrieval from compressed contexts, and compressed long-context QA on LongBench v2 [2]. Across compression budgets, model sizes, and compression methods, KV-CAT consistently improves the quality–budget tradeoff, yielding gains of up to **3.21**$\times$ in suffix perplexity retention, **5**$\times$ in optimization speed, **68%** in retrieval accuracy, and **39%** in long-context QA.

2 KV Cache Compression: Problem Formulation

In this section, we theoretically formalize the KV cache compression problem. A *KV cache compression policy* for a transformer with L layers is a collection of functions $\mathsf{C} = (\mathsf{c}_1, \ldots, \mathsf{c}_L)$, where each c_ℓ maps sequences of n key–value pairs to sequences of length $r(n) \leq n$. For KV pairs in layer ℓ

$$(\boldsymbol{K}, \boldsymbol{V}) = ([\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n]^\top, [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]^\top), \tag{1}$$

we write

$$\mathsf{c}_\ell(\boldsymbol{K}, \boldsymbol{V}) = ([\tilde{\boldsymbol{k}}_1, \ldots, \tilde{\boldsymbol{k}}_{r(n)}]^\top, [\tilde{\boldsymbol{v}}_1, \ldots, \tilde{\boldsymbol{v}}_{r(n)}]^\top), \tag{2}$$

where $r(n)$ is referred to as the *compression budget*. Given a transformer M, a compression policy C, and a sequence of tokens $\boldsymbol{a} = (a_1, \ldots, a_n)$, we define a compressed model $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}$ that shares the parameters of M but uses a modified forward pass. For an input sequence $\boldsymbol{b}$, the computation of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$ proceeds as in $\mathsf{M}([\boldsymbol{a}, \boldsymbol{b}])$, except for the attention over prefix tokens, which is augmented

[1] These methods are also commonly referred to as KV compaction methods in the literature.

the following way: at layer ℓ, let $(\boldsymbol{K_a}, \boldsymbol{V_a})$ denote the KV cache of $\boldsymbol{a}$, based on the original model M and let $\boldsymbol{Y}$ be the representations of $\boldsymbol{b}$ after layer $\ell - 1$ of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}$. We compute

$$\boldsymbol{Q_b} = \boldsymbol{Y}\boldsymbol{W}_Q, \quad \boldsymbol{K_b} = \boldsymbol{Y}\boldsymbol{W}_K, \quad \boldsymbol{V_b} = \boldsymbol{Y}\boldsymbol{W}_V, \tag{3}$$

and replace the prefix KV cache $(\boldsymbol{K_a}, \boldsymbol{V_a})$ with its compressed version $(\tilde{\boldsymbol{K}}_{\boldsymbol{a}}, \tilde{\boldsymbol{V}}_{\boldsymbol{a}}) = \mathsf{c}_\ell(\boldsymbol{K_a}, \boldsymbol{V_a})$. The attention computation then becomes

$$\mathrm{AttnHead}(\boldsymbol{Y}) = \mathrm{softmax}\left(\frac{1}{\sqrt{d_k}}\boldsymbol{Q_b}\begin{bmatrix}\tilde{\boldsymbol{K}}_{\boldsymbol{a}} \\ \boldsymbol{K_b}\end{bmatrix}^\top\right)\begin{bmatrix}\tilde{\boldsymbol{V}}_{\boldsymbol{a}} \\ \boldsymbol{V_b}\end{bmatrix}. \tag{4}$$

Finally, following standard convention in next-token prediction transformers, we define the output of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$ to be the representation of the final token. Thus, compression modifies only the prefix KV pairs, leaving the remainder of the computation unchanged. We now formalize the notion of transformer *KV-compressibility*.

Definition 2.1. Fix $N \in \mathbb{N}$, $\varepsilon > 0$, and let $r : \mathbb{N} \to \mathbb{N}$ be a budget function. A transformer M is said to be (N, ε, r)-compressible[2] if there exists a KV cache compression policy C with budget r such that for every pair of sequences $\boldsymbol{a}$, $\boldsymbol{b}$ of lengths n and k respectively, and with a combined length $n + k \leq N$, it holds that

$$\|\mathsf{M}([\boldsymbol{a}, \boldsymbol{b}]) - \mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})\| < \varepsilon. \tag{5}$$

A more formal treatment of these terms and definitions is provided in Appendix B.1.

3 Motivation and Theoretical Results

Our analysis begins with the following theorem, which shows that almost any sequence-to-vector function can admit transformer implementations with different levels of KV-compressibility. Formal proofs of all results in this section are provided in Appendix B.

Theorem 3.1. *Let A be a finite alphabet, and let $f : \bigcup_{n \leq N} A^n \to \mathbb{R}^{d_{out}}$ be a sequence-to-vector function. Suppose that there exists a sequence $\boldsymbol{a}$ of length n and two sequences $\boldsymbol{b}^1, \boldsymbol{b}^2$ of length $k \leq N - n$ such that*

$$f([\boldsymbol{a}, \boldsymbol{b}^1]) \neq f([\boldsymbol{a}, \boldsymbol{b}^2]). \tag{6}$$

Then for every $\varepsilon > 0$, there exists a transformer that approximates f and is $(N, \varepsilon, 1)$-compressible. Additionally, for every $C > 0$, there exists a transformer of the same architecture, that approximates f but is not (N, C, r)-compressible for any budget function satisfying $r(n) < n$.

Before turning to training methods, we build intuition through a simple motivating example. We show that even for natural sequence-to-vector functions, simple transformer implementations can yield non-compressible representations, while more structured, highly compressible implementations exist.

Motivating example: histogram computation. Let $A = [m]$ be a finite alphabet. Given a sequence $\boldsymbol{a} = (a_1, \ldots, a_n) \in A^n$, the histogram function computes the empirical distribution of symbols,

$$f_{\mathrm{hist}}(\boldsymbol{a}) = \left(\tfrac{n_1}{n}, \ldots, \tfrac{n_m}{n}\right), \tag{7}$$

where $n_i = |\{j : a_j = i\}|$.This is a natural sequence-to-vector function that depends only on aggregate statistics of the input. A straightforward way to implement this function with a 2-layer transformer is as follows. First, take the embedding map to be $\mathrm{emb}(a_i, i) = \boldsymbol{e}_{a_i}$. We take the first transformer layer to be the identity and then apply a second attention layer with uniform attention weights (by setting $\boldsymbol{W}_Q = \boldsymbol{0}$), So that the representation of the final token becomes the average of all previous token embeddings, obtaining

$$\mathsf{M}(\boldsymbol{a}) = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{e}_{a_i} = f_{\mathrm{hist}}(\boldsymbol{a}). \tag{8}$$

This gives a natural implementation of the histogram function with a very simple architecture (see Appendix B.2 for details). Interestingly, we find that this implementation is not compressible:

[2]We include N explicitly, as many natural sequence-to-vector functions require model dimension scaling with N (e.g., $O(N)$ or $O(\log N)$) in order to achieve arbitrarily accurate approximation; see e.g., Sanford et al. [48]; Yehudai et al. [55].

Proposition 3.2. *For any $N \in \mathbb{N}$ and any budget function r with $r(N-1) < N-1$ there exists a constant $C > 0$ such that the above transformer implementation of f_{hist} is not (N, C, r)-compressible.*

Proof sketch. Since $\boldsymbol{W}_O = \mathbf{0}$ in the first layer, compressing the prefix KV cache there has no effect on the suffix representations. In the second layer, $\boldsymbol{W}_Q = \mathbf{0}$, so attention weights are uniform and independent of the keys. Consequently, the compressed model computes the histogram by averaging suffix token embeddings with the compressed value vectors provided by the policy. As compression replaces the prefix by only $r(n)$ vectors, the normalization factor of this average changes. Thus, for any compression policy C,

$$\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a}, \boldsymbol{b}]) = \left(\tfrac{1}{r(n)+k} - \tfrac{1}{n+k}\right) \sum_{i=1}^{k} \boldsymbol{e}_{b_i} + c(\boldsymbol{a}), \tag{9}$$

where $c(\boldsymbol{a})$ depends only on the prefix. The suffix-dependent term can vary over many values as $\boldsymbol{b}$ changes, while $c(\boldsymbol{a})$ is fixed. Therefore, no non-trivial compression policy can uniformly approximate the full model for all suffixes, implying a constant lower bound on the compression error. □

In contrast, a slight modification of the same architecture yields a highly compressible implementation.

Proposition 3.3. *For every $N \in \mathbb{N}$ and $\varepsilon > 0$, there exists a f_{hist} transformer implementation with the same architecture as above that is $(N, \varepsilon, 1)$-compressible.*

The key idea is to leverage positional information, provided by the compression policy, to recover sequence lengths after compression and re-scale the averaged representation so that the correct histogram can be reconstructed from a single KV pair. Importantly, since f_{hist} is permutation-invariant, standard training provides no incentive for the model to preserve positional information, making such compressible solutions unlikely to emerge without additional supervision. This further illustrates that compressibility depends on the learned representation and motivates compression-aware training.

4 Training for KV Cache Compressibility

As shown above, the same task may admit both compressible and non-compressible transformer solutions. This raises the question of whether transformers can be guided toward compressible representations during training. To this end, we introduce KV-Compression Aware Training (KV-CAT), a continued pretraining procedure that explicitly encourages compressible internal representations.

Train-time KV sparsification policy. Our goal is to train transformers whose representations are more amenable to SOTA optimization-based KV cache compression methods. Ideally, one would incorporate such methods directly into the training loop; however, this is computationally prohibitive. A practical alternative is to introduce a simple and scalable training-time KV sparsification policy, such as random KV slot dropping, attention-based filtering, or a lightweight parameterized policy. In our ablations (Appendix F.1), we find that a parameterized policy provides the strongest performance, in terms of both downstream compression and retention of uncompressed accuracy. We hypothesize that this is because adaptive policies better align with the test-time behavior of optimization-based compressors. Accordingly, in our experiments, we implement KV-CAT using lightweight learned routers, described in Appendix C, while emphasizing that the framework is general and can accommodate a range of sparsification policies.

Training objective. We train the transformer using an augmented next-token prediction (NTP) objective. For a sequence $\boldsymbol{a} = (a_1, \ldots, a_n)$, let $p_\theta^{\text{mask}}(\cdot \mid \boldsymbol{a}_{<i})$ denote the model distribution when masking is applied to the KV cache, and $p_\theta^{\text{dense}}(\cdot \mid \boldsymbol{a}_{<i})$ denote the distribution under the standard (unmasked) forward pass. During training, we optimize

$$\mathcal{L}(\theta) = \lambda_{\text{mask}}\mathcal{L}_{\text{mask}} + \lambda_{\text{anchor}}\mathcal{L}_{\text{anchor}} + \lambda_{\text{budget}}\mathcal{L}_{\text{budget}}. \tag{10}$$

The first term is given by

$$\mathcal{L}_{\text{mask}} = \frac{1}{n}\sum_{i=1}^{n} D_{\text{KL}}\big(\text{sg}[p_\theta^{\text{dense}}(\cdot \mid a_{<i})] \,\big\|\, p_\theta^{\text{mask}}(\cdot \mid a_{<i})\big)\,, \tag{11}$$

where sg denotes the stop-gradient operation. This term trains the masked forward pass to match the dense model distribution. The second term $\mathcal{L}_{\text{anchor}} = \frac{1}{n}\sum_{i=1}^{n} -\log p_\theta^{\text{dense}}(a_i \mid a_{<i})$, preserves

Table 1: **KV-CAT perserves performance.** Values are normalized accuracy (%) over 1000 examples.

Model	Variant	HellaSwag	WinoGrande	PIQA	OpenBookQA	ARC-E	ARC-C	Avg.
QWEN2.5-0.5B	Base	**54.6**	52.9	70.4	**38.2**	60.8	32.2	51.5
	KV-CAT	53.6	**54.1**	**72.1**	37.4	**63.6**	**32.4**	**52.2**
QWEN2.5-1.5B	Base	**68.6**	**60.5**	**76.2**	40.0	73.3	**45.2**	**60.6**
	KV-CAT	66.4	**60.5**	75.5	**40.6**	**73.9**	43.5	60.1

the distribution of the dense model. When the KV-sparsification policy is learnable, we also include a budget loss designed to maintain a desirable retention rate, see Appendix C for details. At evaluation time, we use the unmasked forward pass, on top of which we can apply standard KV cache compression methods. The output of KV-CAT is a standard transformer whose representations are trained to be more amenable to compression.

5 Empirical Evaluation

Table 2: **Long-form QA after context compression.** We evaluate the models on seven tasks from LongBench v2. The KV caches of the questions' contexts are compressed with a gradient-based method.

	10% keep		20% keep		50% keep	
Subdomain	Base	KV-CAT	Base	KV-CAT	Base	KV-CAT
Academic	18.1	**26.6**	19.1	**27.7**	21.3	**26.6**
Agent history QA	20.0	**30.0**	25.0	**30.0**	**30.0**	**30.0**
Knowledge graph reasoning	**33.3**	**33.3**	**26.7**	**26.7**	20.0	**26.7**
Legal	27.3	**48.5**	27.3	**45.5**	27.3	**42.4**
Many-shot learning	**28.6**	19.0	**28.6**	**28.6**	**38.1**	33.3
New language translation	**30.0**	**30.0**	15.0	**35.0**	15.0	**35.0**
Table QA	11.1	**27.8**	**22.2**	**22.2**	**38.9**	33.3
Total	22.2	**30.3**	22.2	**30.8**	25.3	**31.2**

We evaluate the effect of KV-CAT training on transformer-based LLMs. Our evaluation considers four settings: (1) standard uncompressed evaluation, to verify that KV-CAT does not degrade base model performance, (2) suffix completion given a compressed prefix, (3) retrieval from a compressed context, and (4) long-context question answering with compressed context. Across experiments, we evaluate two state-of-the-art optimization-based KV compression methods, Attention Matching [61] and a gradient-based optimization method adapted from Eyuboglu et al. [16], on two QWEN2.5 model sizes (0.5B and 1.5B). Overall, KV-CAT preserves model performance in the uncompressed setting, while consistently improving the performance of downstream compression across all tasks and compression budgets, yielding gains of up to **3.21**$\times$ in suffix perplexity retention, **68%** in retrieval accuracy, and **39%** in long-context QA. Results for retrieval from a compressed context are in Table 3, and results for compressed long-context QA are in Table 2. For the full experimental setup and results, see Appendix D, for implementation details, see Appendix E, and for additional results and ablations, see Appendix F.

6 Conclusion

We argue that KV cache compressibility is determined not only by the input, but also by the transformer's learned representation. We prove that almost any sequence-to-vector function admits both highly compressible and inherently non-compressible implementations, motivating compression-aware training. Guided by this perspective, we introduce KV-CAT, a training procedure that encourages compressible representations by exposing the model to KV compression during training. Empirically, models trained with KV-CAT consistently improve the quality–budget trade-off of state-of-the-art post-hoc compression methods across a range of tasks. Overall, our results suggest a complementary approach to post-hoc KV compression: training transformers to become compressible.

Table 3: **Needle retrieval from compressed haystack.** Exact-match accuracy (%) over 100 examples per keep ratio.

	QWEN2.5-0.5B		QWEN2.5-1.5B	
Keep	Base	KV-CAT	Base	KV-CAT
5%	**18**	17	**20**	**20**
10%	12	**15**	**20**	**20**
15%	15	**16**	**21**	**21**
20%	**15**	13	22	**23**
25%	20	**22**	24	**28**
30%	23	**34**	41	**44**
35%	21	**32**	46	**54**
40%	24	**38**	42	**55**
50%	28	**47**	49	**67**
Mean	19.6	**26.0**	31.7	**36.9**

References

[1] Simran Arora and Christopher Ré. Can foundation models help us achieve perfect secrecy? *arXiv preprint arXiv:2205.13722*, 2022.

[2] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664, 2025.

[3] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[4] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239. URL `https://ojs.aaai.org/index.php/AAAI/article/view/6239`.

[5] Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Recurrent memory transformer, 2022.

[6] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic kv cache compression based on pyramidal information funneling, 2025.

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

[8] Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. Towards reasoning era: A survey of long chain-of-thought for reasoning large language models. *arXiv preprint arXiv:2503.09567*, 2025.

[9] Tong Chen, Hao Fang, Patrick Xia, Xiaodong Liu, Benjamin Van Durme, Luke Zettlemoyer, Jianfeng Gao, and Hao Cheng. Generative adapter: Contextualizing language models in parameters with a single forward pass. *arXiv preprint arXiv:2411.05877*, 2024.

[10] Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts, 2023.

[11] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[12] 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`.

[13] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana, 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2097. URL `https://aclanthology.org/N18-2097/`.

[14] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[15] Yam Eitan. The centered convex body whose marginals have the heaviest tails. *arXiv preprint arXiv:2110.14382*, 2021.

[16] Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Ré. Cartridges: Lightweight and general-purpose long context representations via self-study, 2025.

[17] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[18] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[19] Nathan Habib, Clémentine Fourrier, Hynek Kydlíček, Thomas Wolf, and Lewis Tunstall. Lighteval: A lightweight framework for llm evaluation. `https://github.com/huggingface/lighteval`, 2023. GitHub repository.

[20] Alex Horn, Ali Kheradmand, and Mukul Prasad. Delta-net: Real-time network verification using atoms. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 735–749, 2017.

[21] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[22] Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling. *arXiv preprint arXiv:2507.07955*, 2025.

[23] Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. Financebench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944*, 2023.

[24] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models, 2023.

[25] Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing llms in long context scenarios via prompt compression, 2024.

[26] Samuel Karlin and William J Studden. Optimal experimental designs. *The Annals of Mathematical Statistics*, 37(4):783–815, 1966.

[27] Samuel Karlin and William J Studden. Tchebycheff systems: With applications in analysis and statistics. *(No Title)*, 1966.

[28] Samuel Karlin and Zvi Ziegler. Chebyshevian spline functions. *Siam Journal on Numerical Analysis*, 3(3):514–543, 1966.

[29] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[30] Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W Lee, Sangdoo Yun, and Hyun Oh Song. Kvzip: Query-agnostic kv cache compression with context reconstruction. *arXiv preprint arXiv:2505.23416*, 2025.

[31] Junhyuck Kim, Jongho Park, Jaewoong Cho, and Dimitris Papailiopoulos. Lexico: Extreme KV cache compression via sparse coding over universal dictionaries. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 30672–30687, 2025.

[32] 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*, volume 33, pages 9459–9474, 2020.

[33] Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6342–6353, 2023. doi: 10.18653/v1/2023.emnlp-main.391.

[34] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: Llm knows what you are looking for before generation, 2024.

[35] Zhuoling Li, Xiaogang Xu, Zhenhua Xu, SerNam Lim, and Hengshuang Zhao. Larm: Large auto-regressive model for long-horizon embodied intelligence. *arXiv preprint arXiv:2405.17424*, 2024.

[36] Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. Shine: A scalable in-context hypernetwork for mapping context to lora in a single pass. *arXiv preprint arXiv:2602.06358*, 2026.

[37] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[38] CA Micchelli and Allan Pinkus. Moment theory for weak chebyshev systems with applications to monosplines, quadrature formulae and best one-sided l^1-approximation by spline functions with fixed knots. *SIAM Journal on Mathematical Analysis*, 8(2):206–230, 1977.

[39] 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 *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

[40] Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens, 2023.

[41] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

[42] Emre Okcular. Context Engineering - Short-Term Memory Management with Sessions from OpenAI Agents SDK, September 2025.

[43] Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18724–18741, 2024. doi: 10.18653/v1/2024.emnlp-main.1043.

[44] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

[45] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint*, 2019. URL https://arxiv.org/abs/1911.05507.

[46] Prithvi Rajasekaran, Ethan Dixon, Carly Ryan, and Jeremy Hadfield. Effective context engineering for ai agents, September 2025.

[47] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740, 2020. doi: 10.1609/aaai.v34i05.6399. URL https://ojs.aaai.org/index.php/AAAI/article/view/6399.

[48] Clayton Sanford, Daniel J Hsu, and Matus Telgarsky. Representational strengths and limitations of transformers. *Advances in Neural Information Processing Systems*, 36:36677–36707, 2023.

[49] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1454. URL https://aclanthology.org/D19-1454/.

[50] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 47901–47911, 2024.

[51] Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL `https://qwenlm.github.io/blog/qwen2.5/`.

[52] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024.

[53] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. In *International Conference on Learning Representations*, 2025. URL `https://arxiv.org/abs/2410.10819`.

[54] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024.

[55] Gilad Yehudai, Haim Kaplan, Guy Dar, Royi Rassin, Asma Ghandeharioun, Mor Geva, and Amir Globerson. When can transformers count to n? *arXiv preprint arXiv:2407.15160*, 2024.

[56] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

[57] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33: 17283–17297, 2020.

[58] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL `https://aclanthology.org/P19-1472/`.

[59] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models, 2023.

[60] Junhao Zheng, Chengming Shi, Xidi Cai, Qiuke Li, Duzhen Zhang, Chenxing Li, Dong Yu, and Qianli Ma. Lifelong learning of large language model based agents: A roadmap. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

[61] Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast kv compaction via attention matching, 2026.

A Related Work

Token-level context compression and retrieval. One approach to long-context inference reduces the number of input tokens before prompting, rather than modifying the model's hidden state. Retrieval-augmented generation keeps the prompt short by selecting only relevant external text [32], while prompt-compression methods prune or rewrite the provided context in token space [24; 25; 33; 42; 46]. These methods are complementary to latent KV cache compression.

Learned memory and soft-token compression. Another line of work uses learned memory or soft tokens as compact substitutes for long contexts. Recurrent Memory Transformers add memory tokens that carry information between segments [5]; AutoCompressors adapt language models to map earlier segments into summary vectors that act as soft prompts [10]; and gist-tokens train models to compress prompts into reusable learned tokens [40]. DuoAttention instead learns which attention heads require full retrieval over the KV cache, and which can run with a streaming-style state [53]. These methods

modify the model's runtime interface, architecture, or cache policy. Another direction [9; 7; 36] uses hypernetworks to encode long contexts into model parameters, typically via LoRA adapters, rather than storing them in the prompt or KV cache.

Post-hoc KV cache compression. Another line of work focuses on reducing the KV cache of pretrained transformers at inference time. Token-selection and eviction methods keep a subset of original KV slots using attention, recency, or query-dependent importance signals [59; 43; 34; 6; 50]. KVzip selects keys using reconstruction-style objectives over the context [30]. Other post-hoc methods move beyond choosing a subset of the original tokens. Cartridges optimize a compact latent KV cache for each context [16], Attention Matching constructs compact keys and values to preserve attention behavior [61], and Lexico represents KV vectors with sparse codes over learned universal dictionaries [31]. Our work is orthogonal to these compressors: we ask whether the model can be trained so that the same post-hoc methods work better.

B Theory

B.1 Transformers and KV Cache Compression

Notation. For a set A, let A^* denote the set of all finite sequences over A. We denote by $\boldsymbol{e}_i$ the i-th standard (one hot encoded) basis vector. For $n \in \mathbb{N}$, we write $[n] = \{1, \dots, n\}$.

For matrices $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{Y} \in \mathbb{R}^{n \times d'}$, we denote their row-wise concatenation (along the feature dimension) by

$$[\boldsymbol{X}, \boldsymbol{Y}] \in \mathbb{R}^{n \times (d+d')}. \tag{12}$$

For matrices $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{Z} \in \mathbb{R}^{n' \times d}$, we denote their column-wise concatenation (along the sequence dimension) by

$$\begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{Z} \end{bmatrix} \in \mathbb{R}^{(n+n') \times d}. \tag{13}$$

When convenient, we sometimes write $[\boldsymbol{X}^\top, \boldsymbol{Z}^\top]^\top$ instead of $\begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{Z} \end{bmatrix}$.

For vectors $\boldsymbol{v}, \boldsymbol{u} \in \mathbb{R}^n$, we treat them as column vectors in $\mathbb{R}^{n \times 1}$ and write

$$[\boldsymbol{v}, \boldsymbol{u}] \in \mathbb{R}^{n \times 2}. \tag{14}$$

Definition B.1 (Attention Head)**.** Let $d_{\text{in}}, d_{\text{out}} \in \mathbb{N}$. An *attention head* is a function

$$\text{AttnHead} : (\mathbb{R}^{d_{\text{in}}})^* \to (\mathbb{R}^{d_{\text{out}}})^* \tag{15}$$

defined as follows. For a sequence $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]^\top \in \mathbb{R}^{n \times d_{\text{in}}}$,

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_{\boldsymbol{V}}, \tag{16}$$

where $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$. The output is

$$\text{AttnHead}(X) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}, \tag{17}$$

where the softmax is applied row-wise.

Definition B.2 (Transformer Block)**.** Fix $h, d_{\text{in}}, d_{\text{out}}, d_{\text{ff}} \in \mathbb{N}$. A *transformer block* is a function

$$\text{Block} : (\mathbb{R}^{d_{\text{in}}})^* \to (\mathbb{R}^{d_{\text{out}}})^* \tag{18}$$

defined as follows. For a sequence

$$\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]^\top \in \mathbb{R}^{n \times d_{\text{in}}}, \tag{19}$$

we define

$$\text{Block}(\boldsymbol{X}) = \text{FFN}([\text{head}_1(\boldsymbol{X}), \dots, \text{head}_h(\boldsymbol{X})]\boldsymbol{W}_O + \boldsymbol{X}), \tag{20}$$

where each head_i is an attention head with input and output dimensions $d_{\text{in}}, d_{\text{out}}$ respectively, and $\boldsymbol{W}_O \in \mathbb{R}^{h \cdot d_{\text{out}} \times d_{\text{out}}}$, and FFN is a 2-layer feed forward network applied row-wise, i.e.,

$$\text{FFN}(\boldsymbol{X})_i = \text{FFN}(\boldsymbol{x}_i), \quad i = 1, \dots, n, \tag{21}$$

with

$$\text{FFN}(\boldsymbol{x}) = \sigma(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2, \tag{22}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{ff}}}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{out}}}$, $\boldsymbol{b}_1 \in \mathbb{R}^{d_{\text{ff}}}$, and $\boldsymbol{b}_2 \in \mathbb{R}^{d_{\text{out}}}$.

Note. In the theoretical analysis, we omit layer normalization, causal masking, and the residual connection following the FFN for simplicity, as is standard in prior theoretical treatments (See e.g. [48; 55]). Our results extend to the full architecture with only minor modifications. All empirical evaluations are conducted with these components included.

Definition B.3 (Token Embedding)**.** Let A be a finite alphabet. A *token embedding* function is a function

$$\mathrm{emb} : A \times \mathbb{N} \to \mathbb{R}^{d_{\mathrm{model}}}. \tag{23}$$

Given a sequence $\boldsymbol{a} = (a_1, \ldots, a_n) \in A^*$, its embedding is the sequence

$$\mathrm{emb}(\boldsymbol{a}) = [\mathrm{emb}(a_1, 1), \ldots, \mathrm{emb}(a_n, n)]^\top \in \mathbb{R}^{n \times d_{\mathrm{model}}}. \tag{24}$$

Definition B.4 (Transformer)**.** Let A be a finite alphabet. A *transformer* is a tuple

$$\mathsf{M} = (\mathrm{Block}_1, \ldots, \mathrm{Block}_L, \mathrm{emb}), \tag{25}$$

where each $\mathrm{Block}_\ell : (\mathbb{R}^{d_{\ell-1}})^* \to (\mathbb{R}^{d_\ell})^*$ is a transformer block, and $\mathrm{emb} : A^* \to (\mathbb{R}^{d_0})^*$ is a token embedding function.

The transformer M defines a function

$$\mathsf{M} : A^* \to \mathbb{R}^{d_L} \tag{26}$$

as follows. For $\boldsymbol{a} = (a_1, \ldots, a_n) \in A^*$, let

$$\boldsymbol{X}^{(0)} = \mathrm{emb}(\boldsymbol{a}), \qquad \boldsymbol{X}^{(\ell)} = \mathrm{Block}_\ell(\boldsymbol{X}^{(\ell-1)}), \quad \ell = 1, \ldots, L. \tag{27}$$

Then

$$\mathsf{M}(\boldsymbol{a}) = \boldsymbol{X}_n^{(L)}, \tag{28}$$

i.e., the output is the representation of the final token.

Definition B.5 (KV Cache Compression)**.** A *KV cache compression policy* is a tuple $\mathsf{C} = (\mathsf{c}_1, \ldots, \mathsf{c}_L)$, where each

$$\mathsf{c}_\ell : (\mathbb{R}^{d_\ell} \times \mathbb{R}^{d_\ell})^* \to (\mathbb{R}^{d_\ell} \times \mathbb{R}^{d_\ell})^* \tag{29}$$

maps a sequence of key–value pairs to a shorter sequence.

Concretely, for

$$(\boldsymbol{K}, \boldsymbol{V}) = \big([\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n]^\top, [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]^\top\big), \tag{30}$$

we write

$$\mathsf{c}_\ell(\boldsymbol{K}, \boldsymbol{V}) = (\tilde{\boldsymbol{K}}, \tilde{\boldsymbol{V}}) = \big([\tilde{\boldsymbol{k}}_1, \ldots, \tilde{\boldsymbol{k}}_{r(n)}]^\top, [\tilde{\boldsymbol{v}}_1, \ldots, \tilde{\boldsymbol{v}}_{r(n)}]^\top\big), \tag{31}$$

where $r(n) \le n$. The function $r(\cdot)$ is called the *compression budget*.

Given a transformer $\mathsf{M} = (\mathrm{emb}, \mathrm{Block}_1, \ldots, \mathrm{Block}_L)$, a compression policy C, and a *context* sequence $\boldsymbol{a} = (a_1, \ldots, a_n)$, we define the associated *compressed transformer* $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}$.

For an input sequence $\boldsymbol{b} = (b_1, \ldots, b_k)$, the output $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$ is obtained by running the forward pass of M on the concatenated sequence $[\boldsymbol{a}, \boldsymbol{b}] = (a_1, \ldots, a_n, b_1, \ldots, b_k)$, with the following modifications.

For each ℓ and each attention head at the ℓ-th block, after computing keys and values, the compression function c_ℓ is applied to the KV pairs corresponding to the context tokens. The compressed KV pairs replace the original ones in the attention computation.

More precisely, let $(\boldsymbol{K}_{\boldsymbol{a}}, \boldsymbol{V}_{\boldsymbol{a}})$ denote the KV cache produced by the prefix $\boldsymbol{a}$ at some attention head of Block_ℓ of the original model M, and let $\boldsymbol{Y} \in \mathbb{R}^{k \times d_{\ell-1}}$ be the input corresponding to $\boldsymbol{b}$ in the forward pass computation of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}$ after block $\ell - 1$. We first compute

$$\boldsymbol{Q}_{\boldsymbol{b}} = \boldsymbol{Y}\boldsymbol{W}_Q, \quad \boldsymbol{K}_{\boldsymbol{b}} = \boldsymbol{Y}\boldsymbol{W}_K, \quad \boldsymbol{V}_{\boldsymbol{b}} = \boldsymbol{Y}\boldsymbol{W}_V. \tag{32}$$

We then compress the context KV pairs:

$$(\tilde{\boldsymbol{K}}_{\boldsymbol{a}}, \tilde{\boldsymbol{V}}_{\boldsymbol{a}}) = \mathsf{c}_\ell(\boldsymbol{K}_{\boldsymbol{a}}, \boldsymbol{V}_{\boldsymbol{a}}). \tag{33}$$

The resulting attention computation is

$$\mathrm{AttnHead}(\boldsymbol{Y}) = \mathrm{softmax}\left(\frac{1}{\sqrt{d_k}} \boldsymbol{Q}_{\boldsymbol{b}} \begin{bmatrix} \tilde{\boldsymbol{K}}_{\boldsymbol{a}} \\ \boldsymbol{K}_{\boldsymbol{b}} \end{bmatrix}^\top\right) \begin{bmatrix} \tilde{\boldsymbol{V}}_{\boldsymbol{a}} \\ \boldsymbol{V}_{\boldsymbol{b}} \end{bmatrix}. \tag{34}$$

Definition B.6. [KV compressibility] Let $N \in \mathbb{N}$, $\varepsilon > 0$, and let $r : \mathbb{N} \to \mathbb{N}$ be a budget function. A transformer M is said to be (N, ε, r)-compressible if there exists a KV cache compression policy C with budget r such that for every pair of sequences $\boldsymbol{a}, \boldsymbol{b}$ of lengths n and k with combined length satisfying $n + k \leq N$, it holds that

$$\|\mathsf{M}([\boldsymbol{a}, \boldsymbol{b}]) - \mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})\| < \varepsilon. \tag{35}$$

We include N explicitly in the definition above, as many natural sequence-to-vector functions require model dimension scaling with N (e.g., $O(N)$ or $O(\log N)$) in order to achieve arbitrarily accurate approximation; see Sanford et al. [48]; Yehudai et al. [55] for examples.

B.2 Motivating example: histogram computation

In this section we provide formal proofs of Propositions 3.2 and 3.3.

Proof of Proposition 3.2. The construction of $\mathsf{M} = (\mathrm{emb}, \mathrm{Block}_1, \mathrm{Block}_2)$ is straightforward. First, for $j \in [m]$ and $i \in [N]$, define

$$\mathrm{emb}(j, i) = \boldsymbol{u}_j + \boldsymbol{p}_i, \tag{36}$$

where $\boldsymbol{u}_j, \boldsymbol{p}_i \in \mathbb{R}^m$ encode token value and token position respectively, and assume that emb is injective over $[m] \times [N]$.

Let $\boldsymbol{X} \in \mathbb{R}^{n \times m}$. We define $\mathrm{Block}_1(\mathbb{R}^m)^* \to (\mathbb{R}^{2m})^*$ by

$$\mathrm{Block}_1(\boldsymbol{X})_i \approx \rho_1(\boldsymbol{X}_i) \tag{37}$$

where $\rho_1 : \mathbb{R}^m \to \mathbb{R}^{2m}$ is applied row wise $\boldsymbol{X}$ and satisfies for each $j, i \in [m] \times [N]$:

$$\rho_1(\boldsymbol{u}_j + \boldsymbol{p}_i) = \begin{bmatrix} \boldsymbol{e}_j \\ \boldsymbol{0} \end{bmatrix}. \tag{38}$$

Block_1 can be implemented by zeroing out all attention projections and relying on the residual connection together with a feedforward network that approximates ρ_1 to arbitrary precision (this can be obtained as a two-layer feedforward network is a universal approximator of continuous functions on compact sets [14; 21]).

Let $\boldsymbol{X} \in \mathbb{R}^{n \times 2m}$, and write each row as $\boldsymbol{x}_i$. We define

$$\mathrm{Block}_2(\boldsymbol{X})_i = \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j \tag{39}$$

i.e., each output token is replaced by the average of all input tokens. This operation can be implemented using a single attention head by setting

$$\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{0}, \tag{40}$$

so that all attention weights are uniform, and choosing

$$\boldsymbol{W}_V = \boldsymbol{W}_O = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{0} \end{pmatrix}. \tag{41}$$

This choice extracts the first half of each vector and places it in the second half before averaging. After applying a residual connection, the intermediate representation of the i-th token is then

$$\begin{bmatrix} \boldsymbol{x}_i \\ \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{x}_j \end{bmatrix} \tag{42}$$

Finally, the feedforward network is taken to be

$$\mathrm{FFN}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{y}, \tag{43}$$

resulting in a final representation of $\frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i$. For input sequence $\boldsymbol{a}$ of length n we thus have

$$\mathsf{M}(\boldsymbol{a}) \approx \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{e}_{a_i} = \left(\frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{a_i=1},\ \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{a_i=2},\ \ldots,\ \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{a_i=m} \right) = f_{\mathrm{hist}}(\boldsymbol{a}), \tag{44}$$

Thus M approximates f_{hist} to arbitrary precision. Now let $\mathsf{C} = (\mathsf{c}_1, \mathsf{c}_2)$ be any compression policy with budget function $r(\cdot)$ satisfying $r(N-1) < N-1$. Fix a prefix sequence $\boldsymbol{a}$ of length n and a suffix sequence $\boldsymbol{b}$ of length k, and consider the computation of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$.

First, observe that in the first block all attention projection matrices (and in particular $\boldsymbol{W}_O$) are zero. Hence, c_1 has no effect on the representations of the tokens of $\boldsymbol{b}$, and after the first block the representations are exactly as in the uncompressed model.

Next, consider the second block. Let $(\boldsymbol{K}_{\boldsymbol{a}}, \boldsymbol{V}_{\boldsymbol{a}})$ denote the KV cache corresponding to the prefix $\boldsymbol{a}$ computed durning the computation of $\mathsf{M}(\boldsymbol{a})$, and let $(\tilde{\boldsymbol{K}}_{\boldsymbol{a}}, \tilde{\boldsymbol{V}}_{\boldsymbol{a}}) = \mathsf{c}_2(\boldsymbol{K}_{\boldsymbol{a}}, \boldsymbol{V}_{\boldsymbol{a}})$ be the compressed cache. Since $\boldsymbol{W}_Q = 0$, all attention scores are uniform for both M and $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}$ and so

$$\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) = \frac{1}{r(n)+k}\left(\sum_{i=1}^{r(n)} \tilde{\boldsymbol{v}}_i + \sum_{i=1}^{k} \boldsymbol{e}_{b_i}\right). \tag{45}$$

$$\mathsf{M}([\boldsymbol{a},\boldsymbol{b}]) = \frac{1}{n+k}\left(\sum_{i=1}^{n} \boldsymbol{e}_{a_i} + \sum_{i=1}^{k} \boldsymbol{e}_{b_i}\right). \tag{46}$$

For notational convenience, define

$$\tilde{\boldsymbol{v}} = \frac{1}{r(n)+k}\sum_{i=1}^{r(n)} \tilde{\boldsymbol{v}}_i, \quad \tilde{\boldsymbol{a}} = \frac{1}{n+k}\sum_{i=1}^{n} \boldsymbol{e}_{a_i}, \quad \tilde{\boldsymbol{b}} = \sum_{i=1}^{k} \boldsymbol{e}_{b_i}. \tag{47}$$

Then the difference between the compressed and full outputs can be written as

$$\|\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a},\boldsymbol{b}])\| = \left\|\tilde{\boldsymbol{v}} - \tilde{\boldsymbol{a}} + \left(\frac{1}{r(n)+k} - \frac{1}{n+k}\right)\tilde{\boldsymbol{b}}\right\|. \tag{48}$$

We now derive a lower bound. First, choose $\boldsymbol{b} = (1, 1, \ldots, 1)$ and examine the first coordinate. This yields

$$\|\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a},\boldsymbol{b}])\| \geq \left|\tilde{\boldsymbol{v}}_1 - \tilde{\boldsymbol{a}}_1 + \frac{k(r(n)-n)}{(n+k)(r(n)+k)}\right|. \tag{49}$$

Next, choosing $\boldsymbol{b} = (2, 2, \ldots, 2)$ removes the contribution of the last term, giving

$$\|\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a},\boldsymbol{b}])\| \geq |\tilde{\boldsymbol{v}}_1 - \tilde{\boldsymbol{a}}_1|. \tag{50}$$

Combining the two inequalities, we conclude that for any choice of $\tilde{\boldsymbol{v}}$, there exists a $\boldsymbol{b}$ such that

$$\|\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a},\boldsymbol{b}])\| \geq \frac{k(n-r(n))}{2(n+k)(r(n)+k)}. \tag{51}$$

Finally, taking $n = N-1$ and $k = 1$ yields

$$\|\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a},\boldsymbol{b}])\| \geq \frac{N-1-r(N-1)}{2(N)(r(N-1)+1)} = C > 0, \tag{52}$$

which completes the proof. □

Note: for the weakest compression case $r(n) = n-1$, the constant above is $C = O(N^{-2})$, and so as N grows the approximation bound becomes weaker, but for the extreme compression case $r(n) = c$ for some $c \in \mathbb{N}$, C can be chosen independently of N.

proof of Proposition 3.3. We construct $\mathsf{M} = (\text{emb}, \text{Block}_1, \text{Block}_2)$ similarly to the above construction, with a few modifications. Like before, for $j \in [m]$ and $i \in [N]$, define

$$\text{emb}(j, i) = \boldsymbol{u}_j + \boldsymbol{p}_i, \tag{53}$$

where $\boldsymbol{u}_j, \boldsymbol{p}_i \in \mathbb{R}^m$ encode token value and token position respectively, and assume that emb is injective over $[m] \times [N]$.

Let $\boldsymbol{X} \in \mathbb{R}^{n \times m}$. We define $\mathrm{Block}_1(\mathbb{R}^m)^* \to (\mathbb{R}^{4m})^*$ by

$$\mathrm{Block}_1(\boldsymbol{X})_i \approx \rho_1(\boldsymbol{X}_i) \tag{54}$$

where $\rho_1 : \mathbb{R}^m \to \mathbb{R}^{4m}$ is applied row wise $\boldsymbol{X}$ and satisfies for each $j, i \in [m] \times [N]$:

$$\rho_1(\boldsymbol{u}_j + \boldsymbol{p}_i) = \begin{bmatrix} \boldsymbol{e}_j \\ \boldsymbol{p}_i \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}. \tag{55}$$

Where each of the vector blocks above is of size m. That is, Block_1 is identical to the construction in the proof of Proposition 3.2, except that it preserves the positional component. As before, Block_1 can be implemented by zeroing out all attention projections and relying on the residual connection together with a feedforward network that approximates ρ_1 to arbitrary precision. This follows from the universal approximation property of two-layer feedforward networks for continuous functions on compact sets [14; 21].

We construct Block_2 to be composed of a single attention head with

$$\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{0}, \boldsymbol{W}_O = \boldsymbol{I} \tag{56}$$

so that all attention weights are uniform, and choosing

$$\boldsymbol{W}_V = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}. \tag{57}$$

This choice extracts the first m-length block of each vector and places it in the third block before averaging, keeping the first and second vector blocks the same, and zeroing out the last vector block. After applying a residual connection, the intermediate representation of the i-th token a_i is then

$$\begin{bmatrix} \boldsymbol{e}_{a_i} \\ \boldsymbol{p}_i \\ \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{e}_j \\ \boldsymbol{0} \end{bmatrix}. \tag{58}$$

finally, we choose the last feed forward network to approximate a map ρ_2 satisfying for all $i, j, k \in [N]$:

$$\rho_2(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \begin{cases} \boldsymbol{y} & \text{if } \boldsymbol{z} = 0 \\ \frac{j-i+1}{j}\boldsymbol{y} & \text{if } \boldsymbol{z} = \frac{\boldsymbol{p}_i}{k+1} \text{ and } \boldsymbol{x} = \boldsymbol{p}_j. \end{cases} \tag{59}$$

Such a function can be realized to arbitrary precision by a feedforward network: the above specification is defined on a union of disjoint compact sets, and therefore admits a continuous extension to a compact domain, which can in turn be approximated by a standard feedforward network via universal approximation.

Equations 58 and 59 imply that, in the uncompressed setting, for any sequence $\boldsymbol{a}$ of length $n \leq N$, we recover as before

$$\mathsf{M}(\boldsymbol{a}) \approx \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{e}_{a_i} = \left(\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{1}_{a_i=1},\ \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{1}_{a_i=2},\ \ldots,\ \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{1}_{a_i=m} \right) = f_{\mathrm{hist}}(\boldsymbol{a}), \tag{60}$$

and so M approximates f_{hist}. We now define a compression policy $\mathsf{C} = (\mathsf{c}_1, \mathsf{c}_2)$. We first note that, like before, SINCE $\boldsymbol{W}_O = \boldsymbol{0}$ in the first block, c_1 does not effect the forward pass of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}$.

Given a prefix $\boldsymbol{a}$ of length n with KV cache $(\boldsymbol{K_a}, \boldsymbol{V_a})$, we compress it into a single KV pair:

$$\mathsf{c}_2(\boldsymbol{K_a}, \boldsymbol{V_a}) = \Big(\mathbf{0}, \begin{bmatrix} \sum_{i=1}^{n} \boldsymbol{e}_{a_i} \\ \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{p}_n \end{bmatrix}\Big). \tag{61}$$

That is, we store a single value vector containing (i) the unnormalized histogram of the prefix and (ii) its length encoded via $\boldsymbol{p}_n$. Now consider processing a suffix $\boldsymbol{b} = (b_1, \ldots, b_k)$. Since the compressed prefix consists of a single KV entry with zero key, attention again produces a uniform average over the compressed prefix and suffix tokens. The representation of the final token b_k becomes

$$\begin{bmatrix} \boldsymbol{e}_{b_i} \\ \boldsymbol{p}_{n+k} \\ \frac{1}{k+1}\Big(\sum_{i=1}^{n} \boldsymbol{e}_{a_i} + \sum_{i=1}^{k} \boldsymbol{e}_{b_i}\Big) \\ \frac{1}{k+1}\boldsymbol{p}_n \end{bmatrix}. \tag{62}$$

Applying ρ_2 and using equation 59, we obtain

$$\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) \approx \frac{1 + (n+k) - n}{n+k} \cdot \frac{1}{k+1}\Big(\sum_{i=1}^{n} \boldsymbol{e}_{a_i} + \sum_{i=1}^{k} \boldsymbol{e}_{b_i}\Big) = \frac{1}{n+k}\sum_{i=1}^{n+k} \boldsymbol{e}_{c_i}, \tag{63}$$

where c_i ranges over $[\boldsymbol{a}, \boldsymbol{b}]$. This matches the histogram of the concatenated sequence (up to ε), completing the proof.

□

B.3 Compressible Transformers for General Functions

In this section, we present a formal proof of Theorem 3.1. The proof is divided into two lemmas: the first (Lemma B.7) establishes the existence of compressible transformer approximations, and the second (Lemma B.8 demonstrates the existence of non-compressible ones. The theorem then follows immediately.

Lemma B.7. *Let A be a finite alphabet, and let $f : \bigcup_{n \le N} A^n \to \mathbb{R}^{d_{out}}$ be any sequence-to-vector function. Then for every $\varepsilon > 0$ There exists a transformer such that*

1. *(**Approximation**) For every sequence $\boldsymbol{a} = (a_1, \ldots, a_n) \in A^n$ with $n \le N$,*
$$\|f(\boldsymbol{a}) - \mathsf{M}(\boldsymbol{a})\| < \varepsilon. \tag{64}$$
2. *(**Maximal compressibility**) There exists a KV cache compression policy C with compression budget*
$$r(n) \equiv 1 \tag{65}$$
such that for every prefix $\boldsymbol{a} \in A^n$ and suffix $\boldsymbol{b} \in A^k$ with $k + n \le N$,
$$\|\mathsf{M}([\boldsymbol{a}, \boldsymbol{b}]) - \mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})\| < \varepsilon. \tag{66}$$

Proof. We assume that the token embedding map $\mathrm{emb} : A \times \mathbb{N} \to \mathbb{R}^{d_0}$ is defined by

$$\mathrm{emb}(a, i) = \boldsymbol{u}_a + \boldsymbol{p}_i \tag{67}$$

where $\|\boldsymbol{p}_i\| = \|\boldsymbol{p}_j\|$ for all $i, j \in [N]$, and $\boldsymbol{u}_a, \boldsymbol{p}_i \in \mathbb{R}^{d_0}$ for all $a \in A$. We further assume injectivity of the embedding function, that is

$$(a, i) \neq (b, j) \;\Rightarrow\; \mathrm{emb}(a, i) \neq \mathrm{emb}(b, j). \tag{68}$$

Such embeddings can be obtained, for example, using a learned lookup table for tokens combined with a positional encoding (e.g., sinusoidal) with sufficiently large frequencies.

Let $\boldsymbol{u}_\emptyset \in \mathbb{R}^{d_0}$ be a padding vector such that $\boldsymbol{v}_\emptyset \neq \mathrm{emb}(a, i)$ for all $a \in A$ and $i \in \mathbb{N}$. For a sequence $\boldsymbol{a} = (a_1, \ldots, a_n) \in A^*$ with $n \leq N$, define

$$U_{\boldsymbol{a}} = \left\{ \boldsymbol{P}[\mathrm{emb}(a_1, 1), \ldots, \mathrm{emb}(a_n, n), \boldsymbol{u}_\emptyset, \ldots, \boldsymbol{u}_\emptyset]^\top \mid \boldsymbol{P} \in \mathbb{R}^{N \times N} \text{ is a permutation matrix} \right\} \subset \mathbb{R}^{N \times d_0}. \tag{69}$$

That is, we embed the sequence, pad it to length N using $\boldsymbol{u}_\emptyset$, and then apply an arbitrary permutation to the order of the elements.

By injectivity of the embeddings, the sets $\{U_{\boldsymbol{a}} : \boldsymbol{a} \in A^*, |\boldsymbol{a}| \leq N\}$ are pairwise disjoint. Consequently, there exists a continuous function

$$\bar{f} : \mathbb{R}^{N \times d_0} \to \mathbb{R}^{d_{\mathrm{out}}} \tag{70}$$

such that for every $\boldsymbol{a} \in A^*$ with $|\boldsymbol{a}| \leq N$ and every $[\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N]^\top \in U_{\boldsymbol{a}}$,

$$\bar{f}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N) = f(\boldsymbol{a}). \tag{71}$$

Without loss of generality, we may assume that $\bar{f}$ is permutation-invariant. That is, for every permutation $\sigma \in S_N$ and every $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N]^\top \in \mathbb{R}^{N \times d_0}$,

$$\bar{f}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N) = \bar{f}(\boldsymbol{v}_{\sigma(1)}, \ldots, \boldsymbol{v}_{\sigma(N)}). \tag{72}$$

Indeed, if $\bar{f}$ is not permutation-invariant, we can define

$$\tilde{f}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N) = \frac{1}{|S_N|} \sum_{\sigma \in S_N} \bar{f}(\boldsymbol{v}_{\sigma(1)}, \ldots, \boldsymbol{v}_{\sigma(N)}). \tag{73}$$

Then $\tilde{f}$ is continuous and permutation-invariant. Moreover, for every sequence $\boldsymbol{a} \in A^*$ and every $\boldsymbol{V} \in U_{\boldsymbol{a}}$,

$$\tilde{f}(\boldsymbol{V}) = \bar{f}(\boldsymbol{V}), \tag{74}$$

since $U_{\boldsymbol{a}}$ is closed under permutations.

Because $\bar{f}$ is continuous and permutation-invariant, it follows from Zaheer et al. [56] that there exists a pair of functions $\phi : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ and $\rho : \mathbb{R}^{d_1} \to \mathbb{R}^{d_{\mathrm{out}}}$ such that

$$\bar{f}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N) = \rho\left(\sum_{i=1}^{N} \phi(\boldsymbol{v}_i) \right) \tag{75}$$

(Note that some functions require $d_1 = O(N)$). We further assume without loss of generality that $\phi(\boldsymbol{v}_\emptyset) = 0$. We now construct a compressible transformer $\mathsf{M} = (\mathrm{emb}, \mathrm{Block}_1, \mathrm{Block}_2)$ approximating f using this decomposition. The construciton of this transformer closesly resembles the one used in the proof of Proposition 3.3.

For the first block Block_1, we set all attention projection matrices to zero,

$$\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{W}_V = \boldsymbol{W}_O = \boldsymbol{0}, \tag{76}$$

so that the block reduces to its feedforward component.

Let $\tilde{\phi} : \mathbb{R}^{d_0} \to \mathbb{R}^{4d_0}$ be a continuous function satisfying

$$\tilde{\phi}(\mathrm{emb}(a, i)) = \begin{bmatrix} \phi(\mathrm{emb}(a, i)) \\ \boldsymbol{p}_i \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}. \tag{77}$$

for all $a \in A$ and $i \in [N]$ where of the four vector blocks above are of size d_0. That is, $\tilde{\phi}$ computes ϕ on the embedding while preserving positional information and padding with additional zeroes. We choose the feedforward map FFN_1 in this block to approximate $\tilde{\phi}$ to accuracy $\varepsilon' > 0$, to be specified later (This is possible as 2-layer feed forwand networks are universal approximators of continuous functions on compact sets [14; 21]). Thus, for any sequence $\boldsymbol{a} = (a_1, \ldots, a_n)$,

$$\mathrm{Block}_1(\mathrm{emb}(\boldsymbol{a})) \approx \begin{bmatrix} \tilde{\phi}(\mathrm{emb}(a_1, 1)) \\ \vdots \\ \tilde{\phi}(\mathrm{emb}(a_n, n)) \end{bmatrix}. \tag{78}$$

For the second block Block_2, we again use a single attention head, and set

$$\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{0}, \boldsymbol{W}_O = \boldsymbol{I} \tag{79}$$

The value projection $\boldsymbol{W}_V$ is chosen to extract the ϕ-component from the output of $\tilde{\phi}$, and copy it to the third d_1-sized vector block, keep the first $2d_1$ coordinates the same, and zero out the remaining coordinates. That is:

$$\boldsymbol{W}_V = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}. \tag{80}$$

Since $\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{0}$, all attention scores are identical, and hence the attention weights are uniform. Applying the attention update in Block_2 to the outputs of Block_1 and using a residual connection, the representation at position i becomes approximately

$$\begin{bmatrix} \tilde{\phi}(\mathrm{emb}(a_i, i)) \\ \boldsymbol{p}_i \\ \frac{1}{n}\sum_{j=1}^{n} \phi(\mathrm{emb}(a_j, j)) \\ \boldsymbol{0} \end{bmatrix}. \tag{81}$$

We now choose the feedforward network of Block_2 to approximate up to ε'' (to be chosen later) a continuous function $\tilde{\rho}$ satisfying

$$\tilde{\rho}(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \begin{cases} \rho(j \cdot \boldsymbol{y}) & \text{if } \boldsymbol{z} = 0 \text{ and } \boldsymbol{x} = \boldsymbol{p}_j. \\ \rho((j-i+1) \cdot \boldsymbol{y}) & \text{if } \boldsymbol{z} = \frac{\boldsymbol{p}_i}{k+1} \text{ and } \boldsymbol{x} = \boldsymbol{p}_j. \end{cases} \tag{82}$$

Intuitively, this map recovers the position index i from the positional embedding $\boldsymbol{p}_i$, rescales the averaged sum as necessary to recover the unweighted sum, and applies ρ. In particular, at the final position n, the output of M is approximately

$$\rho\left(n \cdot \frac{1}{n}\sum_{j=1}^{n} \phi(\mathrm{emb}(a_j, j))\right) = \rho\left(\sum_{j=1}^{n} \phi(\mathrm{emb}(a_j, j))\right) = f(\boldsymbol{a}). \tag{83}$$

By choosing $\varepsilon', \varepsilon''$ sufficiently small, it follows that M approximates f to arbitrary precision.

We now show that there exists a compression policy with $r(n) = 1$ that satisfies Equation 66.

Define $\mathsf{C} = (\mathsf{c}_1, \mathsf{c}_2)$ as follows. For any sequence of key–value pairs $(\boldsymbol{K}, \boldsymbol{V})$ with $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]^\top$,

$$\mathsf{c}_1(\boldsymbol{K}, \boldsymbol{V}) = (\boldsymbol{0}, \boldsymbol{0}), \tag{84}$$

and

$$\mathsf{c}_2(\boldsymbol{K}, \boldsymbol{V}) = (\tilde{\boldsymbol{K}}, \tilde{\boldsymbol{V}}), \tag{85}$$

where $\tilde{\boldsymbol{K}} = \boldsymbol{0}$ and

$$\tilde{\boldsymbol{V}} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} \sum_{i=1}^{n} \boldsymbol{v}_i \; + \; \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix} \boldsymbol{v}_n. \tag{86}$$

clearly $r(n) = 1$.

For a prefix $\boldsymbol{a}$ of length n and a suffix $\boldsymbol{b}$ of length k, we begin by analyzing the forward pass of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$. After the first block, the representations of the tokens b_i are given by

$$\begin{bmatrix} \tilde{\phi}(\mathrm{emb}(b_1, 1+n)) \\ \vdots \\ \tilde{\phi}(\mathrm{emb}(b_k, k+n)) \end{bmatrix}. \tag{87}$$

This follows because the embedding layer of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}$ is identical to that of M, applied to the concatenated sequence $[\boldsymbol{a}, \boldsymbol{b}]$. Moreover, the computation of Block_1 is applied independently to each token representation, and is therefore unaffected by compression. Next, consider the attention computation in the second block. Since $\boldsymbol{W}_Q = 0$, all attention scores are zero, and hence the attention weights are uniform, equal to $\frac{1}{1+k}$. By the definition of c_2, it follows that after the attention update in Block_2, the representation of each token b_i is

$$\begin{bmatrix} \phi(\mathrm{emb}(b_i, n+i)) \\ \boldsymbol{e}_{n+i} \\ \frac{1}{1+k}\left(\sum_{j=1}^{n} \phi(\mathrm{emb}(a_j, j)) + \sum_{j=1}^{k} \phi(\mathrm{emb}(b_j, n+j))\right) \\ \frac{1}{k+1}\boldsymbol{p}_n \end{bmatrix}. \tag{88}$$

Finally, after applying the feed forward network FFN , by choosing $\varepsilon', \varepsilon''$ small enough, Equation 82 implies that the final output of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$ is approximately

$$\rho\left((k+n-n+1)\cdot\frac{1}{k+1}\big(\sum_{j=1}^{n}\phi(\mathrm{emb}(a_j,j)) + \sum_{j=1}^{k}\phi(\mathrm{emb}(b_j,n+j))\big)\right) \approx f([\boldsymbol{a},\boldsymbol{b}]) \tag{89}$$

and is in an ε approximator of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$, completing the proof. Finally, after applying the feedforward network FFN, and choosing $\varepsilon', \varepsilon''$ sufficiently small, Equation 82 implies that the final output of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$ is approximately

and thus constitutes an ε-approximation of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$, completing the proof.

□

Lemma B.8. *Let A be a finite alphabet, and let $f : \bigcup_{n\le N} A^n \to \mathbb{R}^{d_{out}}$ be any sequence-to-vector function. Assume additionally that for some prefix sequence $\bar{\boldsymbol{a}}$ of length $n < N$ there exists two suffix sequences $\boldsymbol{b}^1$, $\boldsymbol{b}^2$ both of length $k \le N - n$ such that*

$$f([\boldsymbol{a}, \boldsymbol{b}^1]) \neq f([\bar{\boldsymbol{a}}, \boldsymbol{b}^2]) \tag{90}$$

Then for every $\varepsilon, C > 0$ There exists a transformer such that

1. ***(Approximation)** For every sequence $\boldsymbol{a} = (a_1, \ldots, a_n) \in A^n$ with $n \le N$,*

$$\|f(\boldsymbol{a}) - \mathsf{M}(\boldsymbol{a})\| < \varepsilon. \tag{91}$$

2. ***(Non- compressibility)** For every KV cache compression policy C with compression budget satisfying $r(n) < n$*

   there exists a suffix $\boldsymbol{b} \in A^k$ with $k + n \le N$ such that,

$$\|\mathsf{M}([\boldsymbol{a}, \boldsymbol{b}]) - \mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})\| > C. \tag{92}$$

Proof. By the same argument as in the proof of Lemma B.7, there exist functions $\phi : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ and $\rho : \mathbb{R}^{d_1} \to \mathbb{R}^{d_{\mathrm{out}}}$ such that for every $\boldsymbol{a} = (a_1, \ldots, a_n)$,

$$f(\boldsymbol{a}) = \rho\left(\sum_{i=1}^{n} \phi(\mathrm{emb}(a_i, i))\right). \tag{93}$$

Let $\bar{\boldsymbol{a}}$ be the prefix from the theorem statement, and let $\boldsymbol{b}^1, \boldsymbol{b}^2 \in A^k$ be suffixes such that

$$f([\bar{\boldsymbol{a}}, \boldsymbol{b}^1]) \neq f([\bar{\boldsymbol{a}}, \boldsymbol{b}^2]). \tag{94}$$

It follows from equations 90 and 93 that

$$\sum_{i=1}^{k} \phi(\mathrm{emb}(b_i^1, i)) \;\neq\; \sum_{i=1}^{k} \phi(\mathrm{emb}(b_i^2, i)), \tag{95}$$

where $\boldsymbol{b}_1 = (b_1^1, \ldots, b_k^1)$ and $\boldsymbol{b}_2 = (b_1^2, \ldots, b_k^2)$.

Define

$$H = \mathrm{conv}\left(\left\{ \sum_{i=1}^{k} \phi(\mathrm{emb}(b_i, i)) \;\middle|\; (b_1, \ldots, b_k) \in A^k \right\} \right) \;+\; \sum_{i=1}^{n} \phi(\mathrm{emb}(a_i, i)), \tag{96}$$

and let D denote the diameter of H. Since ρ is continuous and H is compact, the image $\rho(H)$ is also compact. Consequently, there exists $\boldsymbol{u}_0 \in \mathbb{R}^{d_{\mathrm{out}}}$ such that

$$\mathrm{dist}(\boldsymbol{u}_0, \rho(H)) > C. \tag{97}$$

We construct the transformer M similarly to the one constructed in the proof of Lemma B.7, with two modifications to the second block.

First, we replace $\boldsymbol{W}_O = \boldsymbol{I}$ with

$$\boldsymbol{W}_O = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}. \tag{98}$$

Second, we modify the second feedforward function to approximate a continuous function $\bar{\rho}$ satisfying

$$\bar{\rho}(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \begin{cases} \rho(i \cdot \boldsymbol{y}), & \text{if } \|\boldsymbol{x} - \boldsymbol{p}_i\| < \varepsilon' \text{ and } i \cdot \boldsymbol{y} \in H_{\frac{\delta}{2}}, \\ \boldsymbol{u}_0, & \text{if } \|\boldsymbol{x} - \boldsymbol{p}_i\| < \varepsilon' \text{ and } i \cdot \boldsymbol{y} \notin H_{\delta}, \end{cases} \tag{99}$$

where

$$\delta = \left(\frac{n+k}{n-1+k} - 1 \right) \frac{D}{2}, \tag{100}$$

and

$$H_\varepsilon := \{ \boldsymbol{z} \in \mathbb{R}^d : \mathrm{dist}(\boldsymbol{z}, H) < \varepsilon \}. \tag{101}$$

By the same reasoning as in Lemma B.7, Equation 91 holds.

Let $\mathsf{C} = (\mathsf{c}_1, \mathsf{c}_2)$ be any compression policy with compression budget $r(n) < n$. Since the first block of M zeroes out all attention heads and depends only on the token embeddings, c_1 has no effect on the forward pass of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$. Thus, after the first block, the representations of the tokens b_i are

$$\begin{bmatrix} \tilde{\phi}(\mathrm{emb}(b_1, 1+n)) \\ \vdots \\ \tilde{\phi}(\mathrm{emb}(b_k, k+n)) \end{bmatrix}. \tag{102}$$

In the second block, since $\boldsymbol{W}_Q = 0$, all attention scores are identical, and hence the attention weights are uniform, equal to $\frac{1}{k+r(n)}$. It follows that the representation of each token b_i after the second block is

$$\bar{\rho}\left(\phi(\mathrm{emb}(\boldsymbol{b}_i, i+n)), \boldsymbol{p}_{i+n}, \frac{1}{r(n)+k} \left(\tilde{\boldsymbol{v}} + \sum_{j=1}^{k} \phi(\mathrm{emb}(b_j, j+n)) \right), \mathbf{0} \right), \tag{103}$$

where the compressed prefix cache is given by

$$\mathsf{c}_2(\boldsymbol{K}_{\boldsymbol{a}}, \boldsymbol{V}_{\boldsymbol{a}}) = (\tilde{\boldsymbol{K}}_{\boldsymbol{a}}, \tilde{\boldsymbol{V}}_{\boldsymbol{a}}) = ([\tilde{\boldsymbol{k}}_1, \ldots, \tilde{\boldsymbol{k}}_{r(n)}]^\top, [\tilde{\boldsymbol{v}}_1, \ldots, \tilde{\boldsymbol{v}}_{r(n)}]^\top) \tag{104}$$

and

$$\tilde{\boldsymbol{v}} = \big(\sum_{i=1}^{r(n)} \tilde{\boldsymbol{v}}_i \big)_{2d_1 : 3d_1}. \tag{105}$$

Since $\frac{n+k}{r(n)+k} > 1$ by Lemma B.9, there exists a sequence $\bar{\boldsymbol{b}}$ of length k such that

$$\operatorname{dist}\left(\frac{n+k}{r(n)+k}\left(\boldsymbol{v}+\sum_{j=1}^{k}\phi(\operatorname{emb}(\bar{b}_j, j+n))\right), H\right) > \left(\frac{n+k}{r(n)+k}-1\right)\frac{D}{2} \;\geq\; \delta. \tag{106}$$

Therefore, by Equation 99, the final representation of the token $\bar{b}_k$, which is the output of $\mathsf{M}_{\mathsf{C},\bar{\boldsymbol{a}}}(\bar{\boldsymbol{b}})$, is equal to $\boldsymbol{u}_0$.

On the other hand, by construction, $\mathsf{M}([\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}]) \in \rho(H)$. Hence,

$$\|\mathsf{M}([\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}]) - \mathsf{M}_{\mathsf{C},\bar{\boldsymbol{a}}}(\bar{\boldsymbol{b}})\| > C, \tag{107}$$

completing the proof.

□

Lemma B.9. *Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in \mathbb{R}^d$ be distinct points with $k \geq 2$, and let*

$$H := \operatorname{conv}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}.$$

Let $\alpha > 1$. Then for every $\boldsymbol{v} \in \mathbb{R}^d$ there exists $i \in [k]$ satisfying

$$\operatorname{dist}(\alpha\boldsymbol{x}_i + \boldsymbol{v}, H) \geq \frac{\alpha-1}{2}\operatorname{diam}(H).$$

Proof. Let

$$D := \operatorname{diam}(H) = \sup_{\boldsymbol{p},\boldsymbol{q}\in H} \|\boldsymbol{p}-\boldsymbol{q}\|.$$

Since the points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ are distinct and $k \geq 2$, we have $D > 0$.

Choose $\boldsymbol{p}, \boldsymbol{q} \in H$ such that $\|\boldsymbol{p}-\boldsymbol{q}\| = D$ (such a pair exists since H is compact), and define

$$\boldsymbol{u} := \frac{\boldsymbol{p}-\boldsymbol{q}}{\|\boldsymbol{p}-\boldsymbol{q}\|}.$$

The set $\{\langle \boldsymbol{u}, \boldsymbol{x}\rangle \mid \boldsymbol{x} \in H\}$ forms a closed interval, and the width of H in direction $\boldsymbol{u}$ is

$$\max_{\boldsymbol{x}\in H}\langle \boldsymbol{u}, \boldsymbol{x}\rangle - \min_{\boldsymbol{x}\in H}\langle \boldsymbol{u}, \boldsymbol{x}\rangle = D.$$

For any $\boldsymbol{v} \in \mathbb{R}^d$, the width of $\alpha H + \boldsymbol{v}$ in direction $\boldsymbol{u}$ is therefore

$$\max_{\boldsymbol{x}\in\alpha H+\boldsymbol{v}}\langle \boldsymbol{u}, \boldsymbol{x}\rangle - \min_{\boldsymbol{x}\in\alpha H+\boldsymbol{v}}\langle \boldsymbol{u}, \boldsymbol{x}\rangle = \alpha D.$$

Let

$$\varepsilon := \frac{\alpha-1}{2}D.$$

Suppose, for contradiction, that

$$\operatorname{dist}(\alpha\boldsymbol{x}_i + \boldsymbol{v}, H) < \varepsilon \qquad \text{for all } i = 1, \ldots, k.$$

Since

$$\alpha H + \boldsymbol{v} = \operatorname{conv}\{\alpha\boldsymbol{x}_1 + \boldsymbol{v}, \ldots, \alpha\boldsymbol{x}_k + \boldsymbol{v}\},$$

and since the ε-neighborhood of H,

$$H_\varepsilon := \{\boldsymbol{z} \in \mathbb{R}^d : \operatorname{dist}(\boldsymbol{z}, H) < \varepsilon\},$$

is convex, it follows that

$$\alpha H + \boldsymbol{v} \subseteq H_\varepsilon.$$

Consequently, the width of $\alpha H + \boldsymbol{v}$ in direction $\boldsymbol{u}$ is strictly smaller than the width of H_ε in direction $\boldsymbol{u}$. This width is at most

$$D + 2\varepsilon.$$

Therefore,

$$\alpha D < D + 2\varepsilon.$$

But by the definition of ε,

$$D + 2\varepsilon = D + (\alpha-1)D = \alpha D,$$

which is a contradiction. Hence there exists some $i \in \{1, \ldots, k\}$ such that

$$\operatorname{dist}(\alpha\boldsymbol{x}_i + \boldsymbol{v}, H) \geq \varepsilon.$$

□

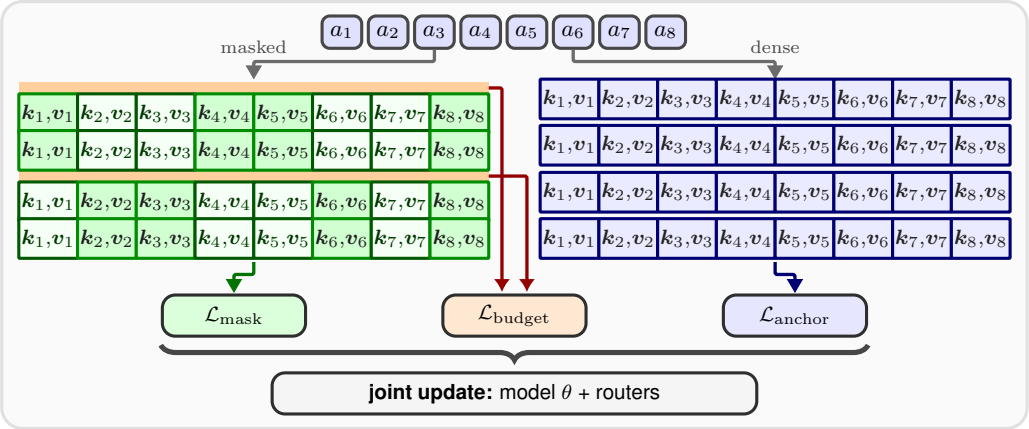


Figure 1: **KV-Compression Aware Training (KV-CAT).** A context $\boldsymbol{a}$ goes through both masked (left) and dense (right) forward passes. In the masked forward pass, routers (orange) compute masks for groups of consecutive layers, marking KV slots as active (green) or inactive (muted green). In the dense forward pass (blue), all KV slots are kept. The output of the masked forward pass is used to compute $\mathcal{L}_{\text{mask}}$, router distributions are used to compute $\mathcal{L}_{\text{budget}}$, and the outputs of the dense forward pass are used to compute $\mathcal{L}_{\text{anchor}}$. These are jointly used to update the parameters.

Note. We believe that a finer-grained analysis of transformer compressibility may be possible through the theory of Chebyshev systems [27], a classical framework with broad applications across both pure and applied mathematics [26; 38; 15; 28]. Chebyshev systems provide tools for controlling the zeros, sign changes, and interpolation structure of linear combinations of functions, which makes them a natural candidate for studying the degrees of freedom available in attention-like mixtures of value functions. We leave a systematic development of this connection to future work.

C KV-CAT implementation details

Masked attention forward pass. At each layer, the train-time KV sparsification policy defines which previous KV slots are visible to attention. For layer ℓ, let $m_j^{(\ell)}$ be the most recent mask produced by the sparsification policy. The attention for query token t is evaluated over

$$\mathcal{A}_t^\ell = \{j \le t : m_j^{(\ell)} = 1\} \tag{108}$$

Thus all valid queries are still processed by the transformer, but their access to past KV slots is restricted to the active, unmasked KV slots. The masks are independent across compression points: a token dropped in one layer may be selected again in a later layer. No masking is used at evaluation time in our comparisons.

Routing mechanism. Starting from a pretrained decoder-only transformer, we insert learned *routers* between consecutive layers and train them jointly with the model. At layer ℓ, the router takes as input the token representations from layer $\ell - 1$ and outputs a scalar score in $[0, 1]$ for each token, indicating its importance. These scores are thresholded using a (non-trainable) hyperparameter τ to produce a binary mask: tokens with scores above τ are active in attention, while the rest are masked out. For efficiency, routers are implemented as linear attention modules. Routers are initialized with all tokens active (i.e., all scores are set to 1), so training begins from the standard dense transformer and gradually learns to mask tokens, using a budget loss (defined below). To further reduce overhead, routers are shared across groups of consecutive layers.

Router parameterization. For the main experiments in the paper, we use lightweight learnable router modules, parameterized as linear attention, as our KV sparsification policy. For a sequence $x_{1:T}$, let $h_t^\ell \in \mathbb{R}^d$ denote the hidden state of token t before decoder layer ℓ. We insert routers at a small set of compression layers $\mathcal{C}$ (e.g., for QWEN2.5-0.5B we use $\mathcal{C} = \{1, 7, 13, 19\}$). Each router independently predicts which KV slots are masked for the following group of layers. Given hidden

states $H^\ell = (h_1^\ell, \dots, h_T^\ell)$ and, the router first computes

$$\tilde{h}_t = \mathrm{LN}(h_t^\ell), \qquad q_t = \phi(W_Q \tilde{h}_t), \quad k_t = \phi(W_K \tilde{h}_t), \quad r_t = W_V \tilde{h}_t, \qquad \phi(z) = \mathrm{ELU}(z) + 1. \tag{109}$$

It then forms a causal linear-attention summary without storing a router KV cache,

$$S_t = \sum_{j \le t} k_j r_j^\top, \qquad z_t = \sum_{j \le t} k_j, \qquad a_t = W_O \frac{q_t^\top S_t}{q_t^\top z_t + \epsilon}. \tag{110}$$

The keep probability is computed from a cosine score between a pointwise projection of the current state and the state after the router summary:

$$u_t = \frac{W_P h_t^\ell}{\|W_P h_t^\ell\|_2}, \qquad w_t = \frac{h_t^\ell + \alpha a_t}{\|h_t^\ell + \alpha a_t\|_2}, \qquad p_t = \frac{1 - \langle u_t, w_t \rangle}{2}. \tag{111}$$

The binary routing decision is

$$m_t^\ell = \mathbf{1}\{p_t > \tau\}, \tag{112}$$

with $\tau = 0.5$ in all reported runs. During training, this hard threshold is optimized with a straight-through estimator: the forward pass uses the binary mask above, while the backward pass passes gradients through the pre-threshold keep probability p_t to the router parameters. We initialize $W_P = -I$ and $\alpha = 0$, which gives $p_t = 1$ for every valid token, so training starts from the dense model.

Training objective. Let $p_\theta^{\mathrm{mask}}(\cdot \mid x_{<t})$ denote the masked forward pass and $p_\theta^{\mathrm{dense}}(\cdot \mid x_{<t})$ the same model with masking disabled. We optimize

$$\begin{aligned} \mathcal{L} = \lambda_{\mathrm{mask}} \frac{1}{T-1} \sum_{t=2}^{T} D_{\mathrm{KL}}\big(\mathrm{sg}\big[p_\theta^{\mathrm{dense}}(\cdot \mid x_{<t})\big] \,\big\|\, p_\theta^{\mathrm{mask}}(\cdot \mid x_{<t})\big) \\ + \lambda_{\mathrm{anchor}} \frac{1}{T-1} \sum_{t=2}^{T} -\log p_\theta^{\mathrm{dense}}(x_t \mid x_{<t}) + \lambda_{\mathrm{budget}} \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \mathcal{B}_c. \end{aligned} \tag{113}$$

where sg stops gradients through the dense teacher. When using the router based KV sparsification policy, we use $\lambda_{\mathrm{budget}} \neq 0$ and the budget term follows the load-balancing objective from Hwang et al. [22]. For target keep rate ρ, define

$$F_c = \frac{1}{T} \sum_{t=1}^{T} m_t^c, \qquad G_c = \frac{1}{T} \sum_{t=1}^{T} p_t^c, \qquad \mathcal{B}_c = \frac{F_c G_c}{\rho} + \frac{(1 - F_c)(1 - G_c)}{1 - \rho}. \tag{114}$$

All transformer and router parameters are updated during continued pretraining; in the reported runs $\rho = 0.5$, $\lambda_{\mathrm{mask}} = 1$, $\lambda_{\mathrm{anchor}} = 1$, and $\lambda_{\mathrm{budget}} = 0.1$.

D Main Experiment Overview

We evaluate the effect of KV-CAT on the performance of downstream optimization-based KV cache compression techniques. Our empirical study is guided by four central questions: **(Q1)** Does KV-CAT training preserve base model performance in uncompressed settings? **(Q2)** Does KV-CAT improve compression quality under a fixed KV-optimization budget? **(Q3)** Does KV-CAT improve retrieval from a KV-compressed context? **(Q4)** Do these gains transfer to KV-compressed long-context question answering? Across all experiments, we compare the original pretrained model to the model obtained via KV-CAT continued pretraining, applying the same downstream compression method at matched KV retention and optimization budgets. We provide full experimental details in Appendix E, and additional results in Appendix F.

KV-CAT training setup. We apply KV-CAT to QWEN2.5-0.5B and QWEN2.5-1.5B checkpoints via continued pretraining on FineWeb-Edu [44]. In both cases, we train all model parameters, and introduce four learned routers shared across four layer groups. Each model is trained on a total of 5.24×10^9 tokens with a max learning rate of 10^{-4}. See further details in Appendix E.1.

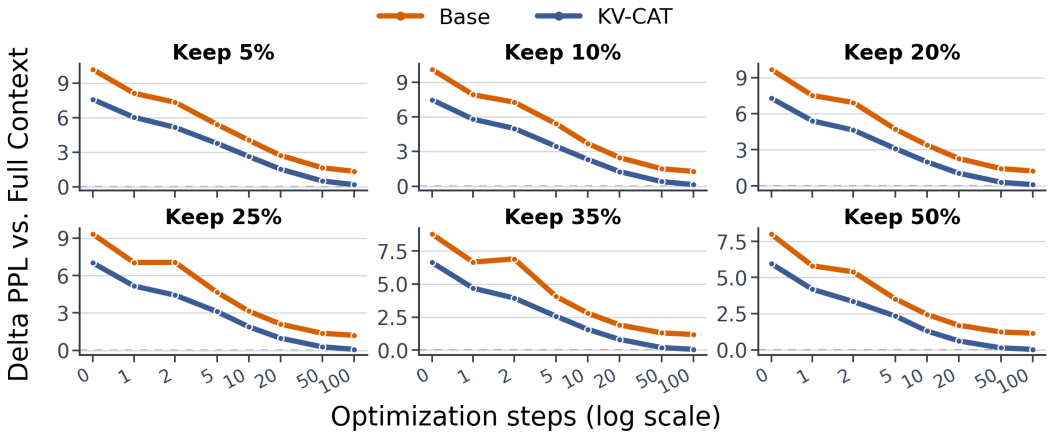


Figure 2: **KV-CAT speeds up gradient-based KV cache compression.** We plot the gap in suffix perplexity under full/compressed-prefix inference throughout gradient-based KV cache optimization. Each panel fixes a different KV keep ratio. Across ratios, the KV-CAT checkpoint achieves a comparable ΔPPL in fewer optimization steps than the base model, yielding up to a **5**× speedup.

KV cache compression methods. Across experiments, we use two post-hoc KV cache compression procedures: Attention Matching [61] and a gradient-based KV cache optimization method adapted from Eyuboglu et al. [16]. Attention Matching constructs a compact prefix cache layer-by-layer, approximating the dense model's attention traces. The gradient-based method instead directly optimizes the compact prefix KV cache to match the dense model's logits on suffix tokens. Both methods compress the KV cache of a given prefix `<text>` by leveraging supervision from a suffix, which together form a *reconstruction sequence*. For the retrieval and QA evaluations, we follow Zweiger et al. [61], and use a reconstruction sequence of the form `<text><instruction><text>`, where `<instruction>` is set to "`Reproduce the preceding passage verbatim.`", and the loss is applied only on the second repeated `<text>` span. Additional implementation details are given in Appendix E.

Preserving uncompressed performance (Q1). We evaluate the effect of KV-CAT on the performance of the model *with no KV cache compression applied*. We compare the base model and the KV-CAT-trained checkpoint on six standard multiple-choice benchmarks using the LightEval harness [19]. As shown in Table 1, KV-CAT closely matches the base model, with an average gain of 0.7 accuracy points for QWEN2.5-0.5B and a drop of 0.5 points for QWEN2.5-1.5B. These results indicate that KV-CAT preserves standard dense behavior rather than trading it off for compressibility.

Compression under fixed KV-optimization budget (Q2). We evaluate compression quality at different keep ratios, under a fixed KV-optimization budget. To do this, we sample held–out prefix–suffix pairs from FineWeb and directly optimize a compacted prefix KV cache to produce the corresponding suffix. We optimize the KV slots via Attention Matching using 256 query states and 8 NNLS iterations across all keep ratios. We report three different similarity metrics between the suffix next token distribution under the full/compressed prefix, averaged over 128 pairs per keep ratio. As seen in Table 4, under a fixed optimization budget, the KV-CAT-trained model consistently outperforms the base model across compression ratios, model sizes, and similarity measures, and achieves up to **3.21**× better retention of suffix perplexity. We additionally evaluate gradient-based KV cache optimization on the same prefix–suffix pairs. As seen in Figure 2, the KV-CAT-trained model achieves better suffix perplexity retention throughout optimization, requiring up to **5**× fewer optimization steps to match the base model's performance. For additional details, see Appendix E.3 and for cross-domain evaluation see Appendix F.2.

Retrieval from a compressed context (Q3). We conduct a needle-in-a-haystack experiment in which we first optimize a compact KV cache for the haystack prefix, and then evaluate exact-match passkey retrieval from the resulting compressed cache. We compress the haystack using the gradient-based method

Table 4: **Suffix completion under prefix compression.** Each example uses a 768-token prefix and a 256-token suffix. Prefix KVs are compressed using Attention Matching. We report: (1) ΔPPL, the difference in suffix perplexity under the full prefix vs. compressed prefix; (2) KL, the divergence between the two token distributions; and (3) Top-1, percentage of tokens on which the two distributions' top-1 agree.

Model	Keep	Variant	ΔPPL (↓)	KL (↓)	Top-1 (↑)
QWEN2.5-0.5B	5%	Base	0.780	0.0562	88.5%
		KV-CAT	**0.461**	**0.0385**	**91.1%**
	10%	Base	0.662	0.0483	89.3%
		KV-CAT	**0.422**	**0.0321**	**91.4%**
	20%	Base	0.777	0.0549	88.7%

described earlier, training the compacted KVs for 100 steps on the reconstruction sequence. We find that KV-CAT improves retrieval accuracy over the base model, particularly at moderate compression budgets. KV-CAT increases mean retrieval accuracy by 6.4 points for QWEN2.5-0.5B and 5.2 points for QWEN2.5-1.5B (Table 3), achieving an 11–19 point improvement at 30–50 percent keep ratios. See additional details in Appendix E.4.

Compressed long-context question answering (Q4). We evaluate on seven LongBench v2 [2] tasks, where the context of each question is compressed using the reconstruction sequence before model prediction. As shown in Table 2, compared to the base model, the KV-CAT-trained model achieves better average accuracy across all KV retention ratios, with an average improvement of up to **39%**. These results demonstrate that KV-CAT yields consistent downstream gains, even when applied to LLMs using only an NTP objective and no post-training. See additional details in Appendix E.5.

E Extended Experimental Details

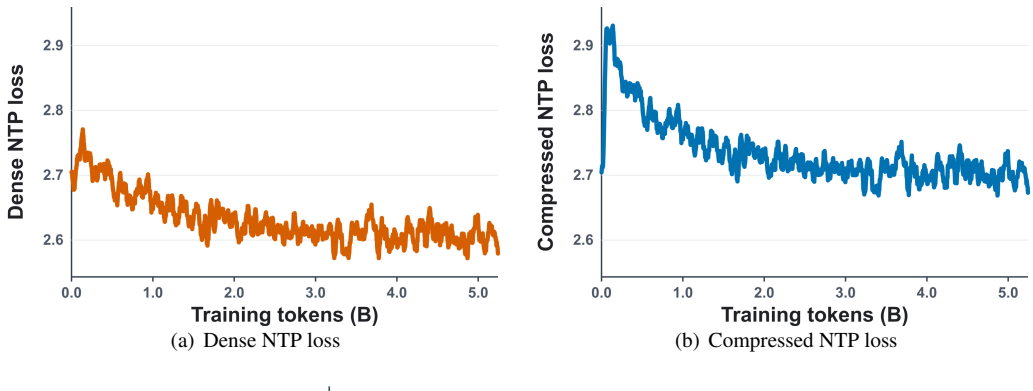
We now describe additional details for the empirical evaluation described in the previous section. We describe the continued-pretraining setup, including the data, architecture hyperparameters, and compression objective, and then give additional details on the post-hoc compression evaluations and experimental setups used throughout the paper. Unless otherwise stated, evaluations compare the base model to the KV-CAT checkpoint with masking disabled. All experiments were run on up to 8 H100 GPUs.

Table 5: **KV-CAT continued-pretraining hyperparameters for QWEN2.5 checkpoints.**

Setting	QWEN2.5-0.5B	QWEN2.5-1.5B
Base checkpoint	`Qwen/Qwen2.5-0.5B`	`Qwen/Qwen2.5-1.5B`
Training data	FineWeb-Edu train, streaming	FineWeb-Edu train, streaming
Sequence length	1024 tokens	1024 tokens
Compression layers	0, 6, 12, 18	0, 7, 14, 21
Router feature dimensions	64	64
Train threshold τ_{train}	0.5	0.5
Target keep rate	50%	50%
Compression loss weight λ_{mask}	1	1
Dense anchor weight λ_{anchor}	1	1
Budget weight λ_{budget}	0.1	0.1
Optimizer	AdamW	AdamW
Weight decay	0.01	0.01
Gradient clipping	1.0 max norm	1.0 max norm
Peak learning rate	10^{-4}	10^{-4}
Minimum learning rate	5×10^{-6}	5×10^{-6}
Warmup steps	600	600
Tokens per optimizer step	131,072	131,072
Training hardware	8×H100 GPUs	8×H100 GPUs
Training steps	40k	40k
Training tokens	5.24×10^9	5.24×10^9

E.1 Continued pretraining runs

We train two model sizes, initialized from the public QWEN2.5-0.5B and QWEN2.5-1.5B [51] checkpoints, using continued pretraining on FineWeb-Edu [44] at a context length of 1024. The compression-aware objective is the one described in the method section: a compressed self-distillation



(a) Dense NTP loss (b) Compressed NTP loss

(c) KV retention ratio

Figure 3: **QWEN2.5-0.5B KV-CAT continued pretraining run.** We plot dense validation next-token-prediction loss, compressed-path validation next-token-prediction loss, and the realized KV retention ratio over the 5.24B-token KV-CAT continued-pretraining run. The retention trace reports the fraction of KV slots kept by the learned routers, whose budget target is 50%.

loss matched to the model's dense distribution, an uncompressed next-token prediction anchor, and a budget penalty. All transformer and compression module parameters are updated.

The router layers are inserted at layers 0, 6, 12, and 18 for QWEN2.5-0.5B and layers 0, 7, 14, and 21 for QWEN2.5-1.5B. Each module is a causal linear-attention boundary predictor (as described in Appendix C) with 64 feature dimensions, a threshold $\tau = 0.5$, and a target keep rate of 50%. The objective weights are $\lambda_{\text{mask}} = 1$, $\lambda_{\text{anchor}} = 1$, and $\lambda_{\text{budget}} = 0.1$. We use AdamW with peak learning rate 10^{-4}, 600 warmup steps, minimum learning rate 5×10^{-6}, weight decay 0.01, and gradient clipping at norm 1.0. Both model sizes use a batch of 131,072 tokens per optimizer step, corresponding to 128 sequences of length 1024, and are trained on 8 GPUs. Runs are configured for 40k optimizer steps, which see 5.24×10^9 tokens.

At evaluation time, the boundary predictors are turned off, and the post-hoc compressor named in each experiment is the only cache-reduction procedure whose quality is measured. Figure 3 reports the compressed and uncompressed validation NTP loss as well as the KV retention ratio throughout the KV-CAT continued pretraining for QWEN2.5-0.5B. Table 5 reports all hyperparameters and training setup for both runs.

E.2 No compression QA evaluation

To check that our training procedure does not substantially degrade ordinary model behavior, we evaluate normalized multiple-choice accuracy on a variety of question answering benchmarks using the LightEval harness [19]. The evaluation suite contains HellaSwag [58], WinoGrande [47], PIQA [4], Social IQa [49], OpenBookQA [39], ARC-Easy and ARC-Challenge [12]. We use 1000 validation examples per task, except for OpenBookQA, whose validation split contains 500 examples in this setup. The reported average is the unweighted mean over these eight tasks. The purpose of this evaluation is to measure whether the trained checkpoints maintain language modeling performance

on standard short-context multiple-choice tasks before we test their behavior under hoc KV cache compression.

E.3 Suffix perplexity under prefix KV cache compression

The suffix-prediction experiments evaluate whether the continued-pretrained checkpoints are easier targets for post-hoc prefix KV cache compression. The in-domain evaluation data consists of held-out blocks from FineWeb. Each evaluation example is a 1024-token excerpt split into a 768-token prefix and a 256-token suffix. We first run the model on the full 1024-token sequence and record the suffix logits and per-layer attention traces. The first 768 tokens define the prefix cache to be compressed. The following 256 suffix tokens are never removed or compressed: after a compact prefix cache has been constructed, the same suffix tokens are appended normally and scored under the original transformer layers. This protocol is shared by the Attention Matching table and the gradient based optimization step-curve figures, but the two experiments fit the compact cache differently.

Attention-Matching. We use the AM-HighestAttnKeys Attention matching variant from Zweiger et al. [61]. For a keep ratio $\rho \in \{0.05, 0.1, 0.2, 0.4\}$, Attention Matching constructs a compact prefix cache independently for each layer and key-value head. Let $K, V \in \mathbb{R}^{L \times d}$ denote the dense prefix keys and values for one key-value head, where $L = 768$, and let $Q \in \mathbb{R}^{M \times d}$ denote suffix query states from the corresponding grouped query heads. We subsample at most 256 suffix queries per key-value head. Attention Matching first selects $\lceil \rho L \rceil$ source keys with the largest root-mean-square attention weight over the sampled suffix queries. The selected keys form the compact keys C_1. A scalar log-bias vector β is then fit by two iterations of box-constrained nonnegative least squares so that the compact cache approximately matches the dense attention normalizer. Finally, compact values C_2 are fit by ridge least squares to reconstruct the dense attention outputs,

$$\operatorname{softmax}\left(\frac{QC_1^\top}{\sqrt{d_k}} + \beta\right) C_2 \approx \operatorname{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V.$$

We use ridge coefficient 10^{-4} with spectral scaling and solve the value regression with a least-squares solver. After constructing the compact prefix cache, we run the model on the same prefix-suffix sequence while replacing the dense prefix cache with the compact cache for all subsequent suffix positions. We report the increase in suffix perplexity relative to the same model's dense-prefix forward pass, KL divergence to the native dense-prefix logits, and top-1 agreement with the native logits. All metrics are averaged over 128 evaluation examples.

Gradient-based KV cache compression. In this setting, we directly optimize a small set of continuous key and value vectors for each example while keeping all model parameters fixed. The resulting optimization curves measure how much information about the prefix can be preserved in a compact KV cache when the cache itself is allowed to adapt to the example. The dense model is first run on the full prefix-suffix sequence to produce teacher logits on the suffix. For a target keep ratio

$$\rho \in \{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.5\},$$

we allocate a compact prefix cache with

$$m = \lceil \rho L \rceil$$

KV slots per layer and key-value head, where $L = 768$ is the dense prefix length. The compact cache consists of learnable key and value tensors

$$\widetilde{K}_{\ell,h}, \widetilde{V}_{\ell,h} \in \mathbb{R}^{m \times d}$$

for each layer ℓ and key-value head h. We initialize $\widetilde{K}_{\ell,h}$ and $\widetilde{V}_{\ell,h}$ from the first m dense prefix KV states for that layer and head. The compact cache is optimized per example using the suffix teacher distribution from the dense full-prefix forward pass. Let $z_t^\star$ denote the dense teacher logits at suffix position t, and let $z_t(\widetilde{K}, \widetilde{V})$ denote the logits obtained when the dense prefix cache is replaced by the compact cache. We minimize

$$\mathcal{L}_{\mathrm{KV}} = \frac{1}{|\mathcal{S}|} \sum_{t \in \mathcal{S}} D_{\mathrm{KL}}\Big(\operatorname{softmax}(z_t^\star) \,\Big\|\, \operatorname{softmax}(z_t(\widetilde{K}, \widetilde{V}))\Big),$$

where $\mathcal{S}$ is the set of suffix prediction positions. The optimization updates only the compact KV tensors; model weights are never updated. We use AdamW for 100 gradient steps with learning rate 10^{-2}, weight decay 0, and gradient clipping at norm 1.0. We record intermediate compact caches after steps

$$\{0, 1, 2, 5, 10, 20, 50, 100\}$$

to produce the optimization-step curves. After each recorded optimization step, we evaluate next-token prediction on the same suffix using the compact cache in place of the dense prefix cache. The reported perplexity is computed against the ground-truth suffix tokens.

This evaluation should be interpreted as a controlled probe of the cache-construction problem in post-hoc compression. For each example, the compact prefix cache is fit using supervision derived from the same prefix–suffix sequence on which it is evaluated: Attention Matching uses suffix attention/query traces to construct the compact cache, while the gradient-based method uses suffix teacher logits. Thus, the comparison asks whether the trained checkpoints make the per-example compact-cache optimization problem easier under matched cache budgets and matched supervision, rather than testing transfer from a cache fit on one suffix to unseen suffixes (which is tested in the next experiments).

E.4 Needle-in-a-haystack retrieval under KV cache compression

We evaluate retrieval under prefix KV cache compression using a deterministic needle-in-a-haystack task. Each example contains neutral filler text, passkey-like distractor numbers, a single six-digit passkey, and a final query asking for the passkey. The prompt length before the final query is 1024 tokens. The model is evaluated using the compressed prompt and the query.

Example construction. To generate each example, we sample a six-digit string $a \in \{000000, \ldots, 999999\}$, which defines the passkey stored in the prefix and the answer expected at evaluation. For example, if $a = 483920$, the needle inserted into the haystack is

```
Memory record:  special_passkey=483920.
```

The final query is

```
Memory record:  special_passkey=
```

and the answer is exactly

```
483920.
```

The passkey is inserted at relative depths

$$d \in \{0, 0.25, 0.5, 0.75, 1\},$$

where $d = 0.0$ places the needle at the beginning of the haystack and $d = 1$ places it at the end. In addition to the answer passkey, examples contain distractors: passkey-like six-digit strings that are explicitly not equal to the answer. These distractors are inserted into the sequence so that the model cannot solve the task by copying an arbitrary number. For example, if the answer is 483920, a possible distractor sentences is or

```
Ignore the misleading special passkey candidate 672041.
```

During example construction, the haystack is assembled from filler text before and after the needle. At each filler-sentence draw, with probability 0.35 we insert a distractor sentence; otherwise we insert a neutral sentence unrelated to the retrieval key. Distractor codes are sampled uniformly from the six-digit code space and rejected if they match the answer. The needle is then inserted at the requested relative depth in the haystack, and the query is appended after the haystack. For each compression ratio, we evaluate 20 instances per depth totaling at a 100 examples.

Training the compact KV cache. For each example, we train a new compact prefix KV cache while keeping all model weights frozen. Let <haystack> denote the haystack prefix, including the filler text, needle and any distractors, but excluding the final query and answer. We construct an reconstruction sequence

```
<haystack><instruction><haystack>,
```

where the instruction is

```
Reproduce the preceding passage verbatim.
Preserve every word, digit, and punctuation mark.
```

The compact cache replaces the KV states for the first `<haystack>`. The loss is applied only on the second `<haystack>`, so the optimizer must store enough information in the compact cache to reconstruct the original prefix, including the passkey and distractors.

For a keep ratio ρ, the compact cache contains

$$m = \lceil \rho L \rceil$$

KV slots per layer and KV head, where L is the haystack length. The compact keys and values are initialized by selecting m random haystack KV states and then optimized directly with AdamW. As in Appendix E.3, we use a KL objective between the suffix logits (computed on the second copy of `<haystack>`) produced by the dense model and the ones produced when attending to the compact cache. For each example, this objective is optimized by running AdamW for 100 steps with learning rate 10^{-2}, weight decay 0, and gradient clipping at norm 1.0.

Evaluation. At evaluation time, the haystack prefix is represented by the optimized compact KV cache, while the final query is appended normally and remains uncompressed. We then perform greedy answer generation and compare the generated completion against the six-digit passkey. In the table, we report native-success-conditioned exact-match retrieval accuracy: for each model, keep ratio, and compact-cache optimization step, we report the percentage of those examples for which the compressed-prefix model also generates exactly the passkey.

E.5 Long-form question answering under KV cache compression

We evaluate long-context question answering under post-hoc KV cache compression on LongBench v2. Because the compression procedure optimizes a separate compact KV cache for each prompt, this evaluation is substantially more expensive than ordinary decoding. We therefore report results on a fixed subset of seven LongBench v2 subdomains: Academic, Agent history QA, Knowledge graph reasoning, Legal, Many-shot learning, New language translation, and Table QA. This subset contains 221 examples in total.

For each LongBench example, we construct a multiple-choice prompt from the context, question, and four answer choices. The prefix is

```
Please read the following text and answer the question below.
```

followed by the context from the benchmark. The suffix is

```
What is the correct answer to this question: <question>
```

followed by the four choices `(A)`, `(B)`, `(C)`, and `(D)`, and the answer-format prefix

```
The correct answer is (.
```

We score the four continuations `A)`, `B)`, `C)`, and `D)` by total log-likelihood and predict the answer with the highest score. No free-form generation is used.

For each example, model, and keep ratio $\in \{0.1, 0.2, 0.5\}$, we optimize a compact KV cache for example's context while keeping all model weights frozen. The optimization uses a reconstruction sequence as in Appendix E.4. As common in LongBench evaluations, if the reconstruction sequence length is larger than the context length of the model (32,768 tokens) we truncate the context using middle truncation (keeping a prefix and suffix of equal length) [2]. The KC objective is a KL divergence between the dense model's next-token distribution and the compact-cache model's next-token distribution on all tokens of the repeated prefix. The compressed caches are initialized from the first prefix KV slots. We optimize each compact cache for 100 AdamW steps with learning rate 10^{-2}, weight decay 0, gradient clipping norm 1.0, and gradient checkpointing enabled. At evaluation time, the optimized compact KV cache replaces the full KV cache of the context prefix. The question, answer choices, and answer prefix are then processed normally, without compression. We report accuracy, grouped by subdomain and averaged over the 221 examples.

Table 6: KV-CAT implemented with a ROUTER policy retains base model performance better than the fixed RAND and ATTN policies.

Model	Variant	HellaSwag	WinoGrande	PIQA	OpenBookQA	ARC-E	ARC-C	Avg.
QWEN2.5-0.5B	Base	**54.6**	52.9	70.4	**38.2**	60.8	32.2	51.5
	ROUTER	53.6	**54.1**	**72.1**	37.4	**63.6**	**32.4**	**52.2**
	RAND	52.6	52.9	70.2	35.4	61.8	31.1	50.7
	ATTN	50.7	53.6	70.0	35.6	60.2	30.9	50.2

F Additional Experimental Results

This section collects supplementary results that are complementary to Section 5. In Section F.1, we ablate the choice of the train-time sparsification policy used in KV-CAT. In Section F.2, we test whether the Attention Matching performance gains we observe in Section 5 generalize to prefix–suffix from additional text corpora.

F.1 Train-time KV sparsification policy ablation

We use this ablation to choose the train-time KV sparsification policy used in the main experiments. Starting from QWEN2.5-0.5B, we train three checkpoints with the same self-distillation and dense-anchor losses and the same 50% target keep rate, varying only the policy that selects active KV slot during the masked forward pass. RAND uniformly samples an exact-count 50% subset of valid KV slots at each sparsification point. ATTN is an H2O-inspired [59] attention-mass baseline: it computes dense causal attention at the routed layer, sums the attention mass received by each source token over heads and valid query positions, and keeps the top 50% source KV slots. ROUTER uses the lightweight learned linear-attention routers described in Appendix C. Because RAND and ATTN satisfy the target keep rate by construction, we set $\lambda_{\text{budget}} = 0$ for those runs; for ROUTER, we use the budget regularizer described in Appendix C. In all comparisons below, the trained checkpoints are evaluated in the unmasked forward pass mode, with the training-time sparsification mechanism disabled, and post-hoc compression is applied afterward.

Uncompressed performance. ROUTER is the only sparsification policy in this ablation that preserves dense downstream accuracy at or above the base model average. Its average score is 52.2, compared to 51.5 for the base model, 50.7 for RAND, and 50.2 for ATTN. The fixed policies still retain much of the base model's zero-shot performance after continued pretraining.

Attention Matching Evaluation. All checkpoints are evaluated with the same Attention Matching procedure described in Appendix E.3. Table 7 shows that all three train-time sparsification policies Attention Matching performance compared to the original base checkpoint for every reported metric and budget. This further supports the KV-CAT framework, demonstrating that the downstream compression improvements are not tied exclusively to learned router parameterization. Among the policies, ROUTER gives the best result for every reported keep ratio and metric. The margin over RAND is sometimes small, especially at 10% and 40% keep ratio, but ROUTER is consistently at least as good under compression and is clearly stronger on dense task retention. We therefore use ROUTER in the main experiments as the default train-time sparsification policy.

Table 7: Attention-matching prefix compression on FineWeb-Edu for the base model and three KV-CAT models with the ROUTER, RAND, and ATTN KV sparsification policies. We report degradation relative to each model's native dense-prefix forward pass. Lower ΔPPL and KL are better; higher top- 1 agreement is better.

Keep ratio	ΔPPL (↓)				KL (↓)				Top-1 (↑)			
	Base	ROUTER	RAND	ATTN	Base	ROUTER	RAND	ATTN	Base	ROUTER	RAND	ATTN
0.05	0.780	**0.461**	0.547	0.593	0.0562	**0.0385**	0.0420	0.0440	88.5%	**91.1%**	90.3%	90.3%
0.10	0.662	**0.422**	0.427	0.495	0.0483	**0.0321**	0.0356	0.0363	89.3%	**91.4%**	90.9%	90.8%
0.20	0.777	**0.438**	0.547	0.594	0.0549	**0.0374**	0.0409	0.0396	88.7%	**90.8%**	90.4%	90.5%
0.40	0.592	**0.360**	0.361	0.386	0.0431	**0.0272**	0.0283	0.0288	90.3%	**92.5%**	92.3%	92.4%

F.2 Cross-domain attention matching evaluation

We test whether the Attention Matching optimization gains observed for KV-CAT transfer to prefix–suffix pair from additional textual domains. We evaluate Attention Matching KV cache compression on validation examples from three held-out corpora: WikiText-103 [37], PG-19 [45], and arXiv [13]. For each example, we take a 768-token prefix and a 256-token suffix, compress only the prefix KV cache with the same Attention Matching procedure used in Appendix E.3, and then score suffix next-token prediction against each model's own dense full-prefix distribution.

Across all three held-out domains, the compression-aware checkpoint remains a better target for the same post-hoc compressor. It reduces KL and increases top-1 agreement for every corpus and keep ratio. It also lowers ΔPPL in 11 of 12 settings; the only exception is arXiv at 40% keep ratio, where ΔPPL increases even though KL and top-1 agreement still improve. We therefore interpret this table as evidence that KV-CAT improves cross-domain compressibility under Attention Matching.

F.3 Additional KV keep-ratio curves

Figure 4 extends the main-body optimization curves in Figure 2 with three additional KV keep ratios: 15%, 25%, and 35%.

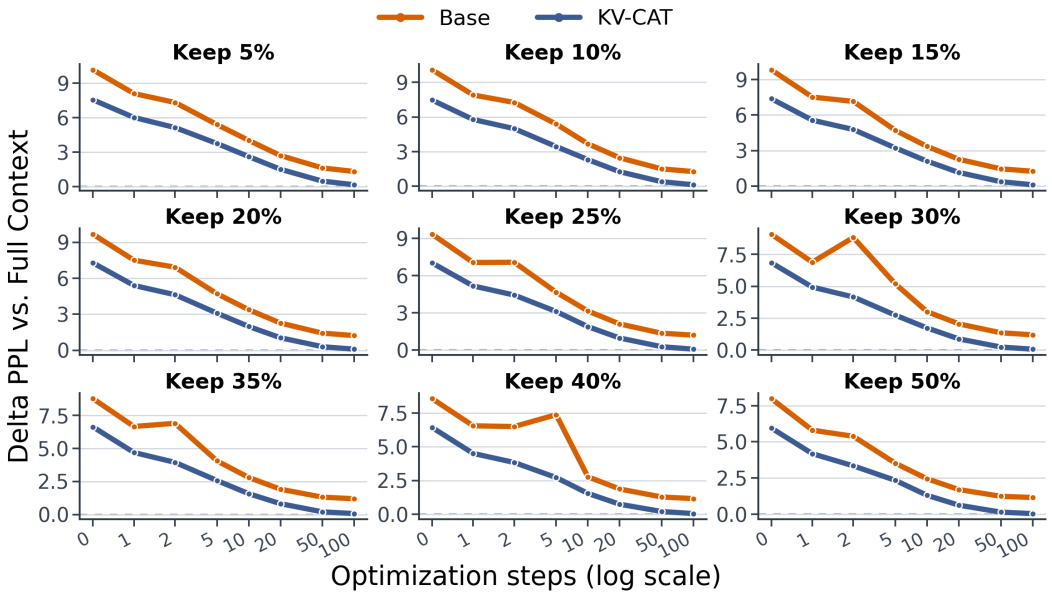


Figure 4: **KV-CAT speeds up gradient-based KV cache compression.** We plot the gap in suffix perplexity under full/compressed-prefix inference throughout gradient-based KV cache optimization. Each panel fixes a different KV keep ratio. Across ratios, the KV-CAT checkpoint achieves a comparable ΔPPL in fewer optimization steps than the base model, yielding up to a **5**$\times$ speedup.