Training Transformers for KV Cache Compressibility

Yoav Gelberg[♠] [*] [†]
gelberg@cs.stanford.edu

Yam Eitan[♣*]
yam.eitan@campus.technion.ac.il

Michael Bronstein[♢] **Yarin Gal**[♢] **Haggai Maron**[♣♡]

[♠]Stanford University [♢]University of Oxford
[♣]Technion – Israel Institute of Technology [♡]NVIDIA

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 [43; 25; 2], to long-form reasoning and continuously learning agents [8; 66; 38]. 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 [65; 37; 57].

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 [31; 11; 59; 60], state space models [19; 18], sparse attention variants [4; 63], and more. These methods significantly reduce the computational cost of

[*]Equal contribution. Order determined by a coin flip.
[†]Work conducted while at the University of Oxford.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

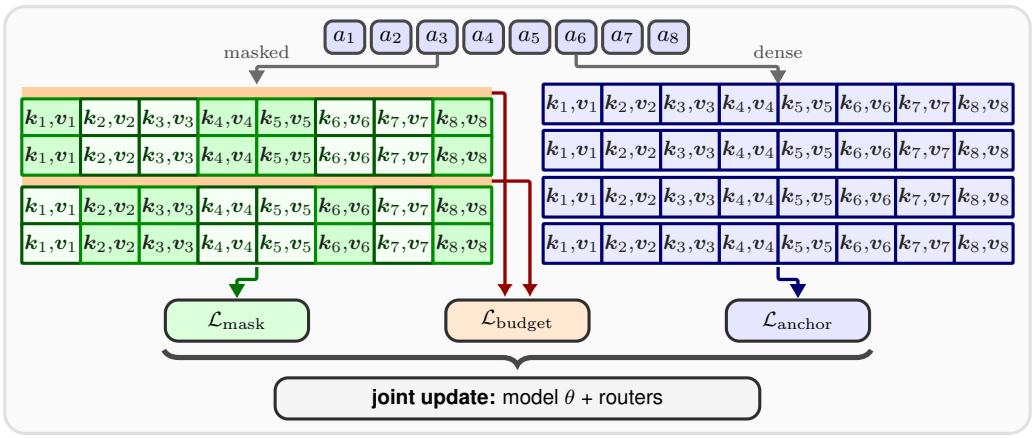


Figure 1: **KV-Compression Aware Training (KV-CAT).** A context $\boldsymbol{a}$ goes through both masked (left) and dense (right) forward passes. In the masked forward pass, routers (orange) compute masks for groups of consecutive layers, marking KV slots as active (green) or inactive (muted green). 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.

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 [50; 44], learned token filtering [26], and policies based on attention patterns, recency, heavy-hitter behavior, or layer-wise importance [65; 57; 37; 7]. 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. [67] 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

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

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.

To demonstrate that KV-CAT improves the quality–budget tradeoff of post-hoc KV cache compression, we apply it to QWEN2.5 models [54], and evaluate state-of-the-art compression methods on the KV caches of the resulting checkpoints. We measure performance along three axes: (i) suffix perplexity under prefix compression, (ii) retrieval accuracy from a compressed prefix, and (iii) compressed long-context question answering on LongBench v2 [3] tasks. Across a range of compression budgets, model sizes, and compression methods, models trained with KV-CAT consistently achieve a better quality–budget tradeoff than the base model, yielding improvements of up to **2.58**$\times$ in suffix perplexity retention, **5**$\times$ in optimization time, **51%** in retrieval accuracy, and **19%** in long-context QA. Importantly, our goal is not to replace post-hoc methods, but to make models more amenable to them, enabling more effective compression.

Contributions. In summary, this paper makes the following contributions:

(1) We characterize KV-compressibility as a property of the learned transformer representation, rather than solely of the task or the input sequence.

(2) We provide theoretical results showing that almost all sequence-to-vector functions can admit both compressible and non-compressible transformer implementations.

(3) We propose KV-CAT, a training procedure that promotes compressible KV representations.

(4) We show that KV-CAT improves post-hoc KV cache compression at matched cache retention rates across: suffix perplexity retention, retrieval, and 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 in the following way: at layer ℓ, let $(\boldsymbol{K}_{\boldsymbol{a}}, \boldsymbol{V}_{\boldsymbol{a}})$ 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

$$\boldsymbol{Q}_{\boldsymbol{b}} = \boldsymbol{Y}\boldsymbol{W}_Q, \quad \boldsymbol{K}_{\boldsymbol{b}} = \boldsymbol{Y}\boldsymbol{W}_K, \quad \boldsymbol{V}_{\boldsymbol{b}} = \boldsymbol{Y}\boldsymbol{W}_V, \tag{3}$$

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

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

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

[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. [52]; Yehudai et al. [61].

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 A.1.

3 Motivation and Theoretical Results

We begin with a theoretical analysis of KV-compressibility, motivated by the following question:

> *Can the same sequence-to-vector function admit transformer implementations with different levels of KV-compressibility?*

Theorem 3.1 answers this affirmatively for almost any sequence-to-vector function. The proof is deferred to Appendix A.3.

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 $\varepsilon, C > 0$:

1. *There exists a transformer that approximates f and is $(N, \varepsilon, 1)$-compressible.*
2. *There exists a transformer of the same architecture, that approximates f but is not (N, C, r)-compressible for any budget function satisfying $r(n) < n$.*

Implication for training. Theorem 3.1 demonstrates that optimizing solely for full-cache predictive accuracy during standard training leaves a critical degree of freedom unaddressed: within a single architecture, equally accurate implementations of the same function can differ drastically in their KV-compressibility. Crucially, this limitation cannot always be resolved by post-hoc compression, as the theorem establishes that certain learned implementations incur constant error under any non-trivial compressor. These theoretical bounds motivate incorporating compressibility directly into the training signal to incentivize representations that maintain predictive quality under compressed caches. While Theorem 3.1 is an existence result and does not guarantee that standard LMs converge to incompressible solutions, our empirical evaluation investigates whether this degree of freedom can be leveraged to improve real-world LM compressibility.

Before introducing our training method, we build intuition through a simple motivating example. We show that for natural sequence-to-vector functions, even simple transformer implementations can yield non-compressible representations, while more structured, highly compressible implementations exist. A more detailed discussion, including full proofs of the results below, is provided in Appendix A.2.

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,

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

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[3] $\text{emb}(a_i, i) = \boldsymbol{e}_{a_i}$. We take the first transformer layer to be the identity [4], 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

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

This gives a natural implementation of the histogram function with a very simple architecture. Interestingly, we find that this implementation is not compressible:

[3] We typically use token embeddings of the form $\text{emb}(a_i, i) = \boldsymbol{u}_{a_i} + \boldsymbol{p}_i$. The embeddings used above can be obtained from this representation by a minor adjustment to the first attention layer; see Appendix A.2 for details.

[4] This can be done by setting $\boldsymbol{W}_O = \boldsymbol{0}$ and choosing the feedforward network to be the identity.

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. Recall that in the compressed setting the input is $[\boldsymbol{a}, \boldsymbol{b}]$ and we want to compress the KV caches associated with the prefix $\boldsymbol{a}$. We begin by observing that since in the first layer $\boldsymbol{W}_O = \mathbf{0}$, compressing the prefix of the first layer has no effect on the representation of the tokens of $\boldsymbol{b}$. Additionally, since in the second layer $\boldsymbol{W}_Q = \mathbf{0}$, the attention scores are independent of the keys, and therefore the attention weights are uniform across all tokens. Thus, for a prefix sequence $\boldsymbol{a}$ of length n and a suffix $\boldsymbol{b}$ of length k, the compressed model satisfies, for any compression policy C,

$$\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) = \frac{1}{r(n)+k}\Big(\sum_{i=1}^{r(n)} \tilde{\boldsymbol{v}}_i + \sum_{i=1}^{k} \boldsymbol{e}_{b_i}\Big). \tag{9}$$

Similarly, the full model computes

$$\mathsf{M}([\boldsymbol{a}, \boldsymbol{b}]) = \frac{1}{n+k}\Big(\sum_{i=1}^{n} \boldsymbol{e}_{a_i} + \sum_{i=1}^{k} \boldsymbol{e}_{b_i}\Big). \tag{10}$$

For notational convenience, define

$$\tilde{\boldsymbol{v}} = \frac{1}{r(n)+k}\sum_{i=1}^{r(n)} \tilde{\boldsymbol{v}}_i, \quad \tilde{\boldsymbol{a}} = \frac{1}{n+k}\sum_{i=1}^{n} \boldsymbol{e}_{a_i}, \quad \tilde{\boldsymbol{b}} = \Big(\frac{1}{r(n)+k} - \frac{1}{n+k}\Big)\sum_{i=1}^{k} \boldsymbol{e}_{b_i}. \tag{11}$$

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

$$\|\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a}, \boldsymbol{b}])\| = \left\|\tilde{\boldsymbol{v}} - \tilde{\boldsymbol{a}} + \tilde{\boldsymbol{b}}\right\|. \tag{12}$$

Both $\tilde{\boldsymbol{v}}$ and $\tilde{\boldsymbol{a}}$ depend only on the prefix $\boldsymbol{a}$, while $\tilde{\boldsymbol{b}}$ depends on the suffix $\boldsymbol{b}$ and can vary over a range of values. For $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$ to ε-approximate $\mathsf{M}([\boldsymbol{a}, \boldsymbol{b}])$ uniformly over all suffixes $\boldsymbol{b}$, Equation 12 requires that the fixed term $\tilde{\boldsymbol{v}} - \tilde{\boldsymbol{a}}$ be ε-close to every possible value of $-\tilde{\boldsymbol{b}}$. For sufficiently small ε, this is impossible. Hence, the error is bounded below by a constant. □

We now show that a slight modification makes the above implementation highly compressible.

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, ε, r)-compressible with $r(n) = 1$.*

Proof sketch. We assume an injective token embedding of the form $\text{emb}(a_i, i) = \boldsymbol{u}_{a_i} + \boldsymbol{p}_i$ where $\boldsymbol{p}_i$ are positional embeddings. As in the previous construction, we set $\boldsymbol{W}_O = \mathbf{0}$ in the first transformer layer, but now choose the feedforward network to approximately satisfy

$$\rho_1(\boldsymbol{u}_{a_i} + \boldsymbol{p}_i) = [\boldsymbol{e}_{a_i}^\top, \boldsymbol{p}_i^\top, \mathbf{0}^\top, \mathbf{0}^\top]^\top \tag{13}$$

That is, each token representation is partitioned into four blocks: the first encodes token identity, the second retains positional information, and the last two are auxiliary slots used later in the construction. We then apply a second attention layer with $\boldsymbol{W}_Q = \mathbf{0}$, so attention is uniform. With the residual connection, the final token representation can be made to contain the empirical average in the third block while preserving the positional encoding $\boldsymbol{p}_n$, which encodes the sequence length. The final feedforward network ρ_2 is designed to behave differently depending on the available information. In the uncompressed setting, the value of the final block remains $\mathbf{0}$, in which case it simply returns the empirical average (from the third slot). In the compressed setting, however, an optimal compression policy can provide additional positional information in the final vector block, which can then be used to correct for the distortion introduced by compression. Specifically, we define a compression policy that maps the prefix to a single KV pair, whose value stores the *unnormalized* histogram together with the prefix length (encoded via $\boldsymbol{p}_n$ in the last block). When processing a suffix, attention averages this summary with the suffix tokens, producing a mis-scaled estimate of the histogram. Crucially, the model now has access to both the original prefix length and the total sequence length (via positional encodings). The final feedforward network uses this information to re-normalize the average and recover the correct histogram. This establishes (N, ε, r)-compressibility with $r(n) = 1$. □

Table 1: **KV-CAT largely preserves base model performance.** We test KV-CAT-trained QWEN2.5 checkpoints on a variety of QA benchmarks without any KV cache compression applied. We report normalized multiple-choice accuracy (%) evaluated on the official evaluation splits.

Model	Variant	HellaSwag	WinoGrande	PIQA	OpenBookQA	ARC-E	ARC-C	Avg.
QWEN2.5-0.5B	Base	**52.19**	**54.22**	69.80	**38.20**	58.46	**32.25**	**50.85**
	KV-CAT	51.09	52.88	**70.78**	37.40	**60.73**	32.08	50.83
QWEN2.5-1.5B	Base	65.16	57.46	75.03	40.20	**71.59**	**44.54**	59.00
	KV-CAT	**65.59**	**58.96**	**75.14**	**40.80**	71.13	44.11	**59.29**

For a formal proof of Proposition 3.3, see Appendix A.2. Note that while incorporating positional information to correct attention weights under compression is conceptually straightforward, a standard transformer trained only on full (uncompressed) sequences is not explicitly encouraged to learn such a mechanism. Since computing the histogram depends only on token identities and is invariant to their positions, the model can, in principle, learn to ignore positional information. This suggests that obtaining compressible solutions may benefit from a training procedure that encourages compression.

4 Training for KV Cache Compressibility

As shown in the previous section, training transformers to solve sequence-to-vector tasks may admit both compressible and non-compressible solutions. This naturally raises the following question:

Can we guide transformers toward more compressible representations during training?

To answer this question, 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 generic KV cache compression methods. Ideally, one would incorporate such methods directly into the training loop, adapting the compression policy as the model evolves; 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 that adapts jointly with the model. In our ablations (Appendix D.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, whereas fixed policies may be too rigid. Accordingly, in our experiments, we implement KV-CAT using lightweight learned routers, described below, while emphasizing that the framework is general and can accommodate a range of sparsification policies.

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 nearly 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. See Appendix B for additional details.

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

$$\mathcal{L}(\theta) = \lambda_{\text{mask}}\mathcal{L}_{\text{mask}} + \lambda_{\text{budget}}\mathcal{L}_{\text{budget}} + \lambda_{\text{anchor}}\mathcal{L}_{\text{anchor}}. \tag{14}$$

The first term is given by

$$\mathcal{L}_{\text{mask}} = \frac{1}{n}\sum_{i=1}^{n} D_{\text{KL}}\big(\text{sg}[p_\theta^{\text{dense}}(\cdot \mid a_{<i})] \,\big\|\, p_\theta^{\text{mask}}(\cdot \mid a_{<i})\big)\,, \tag{15}$$

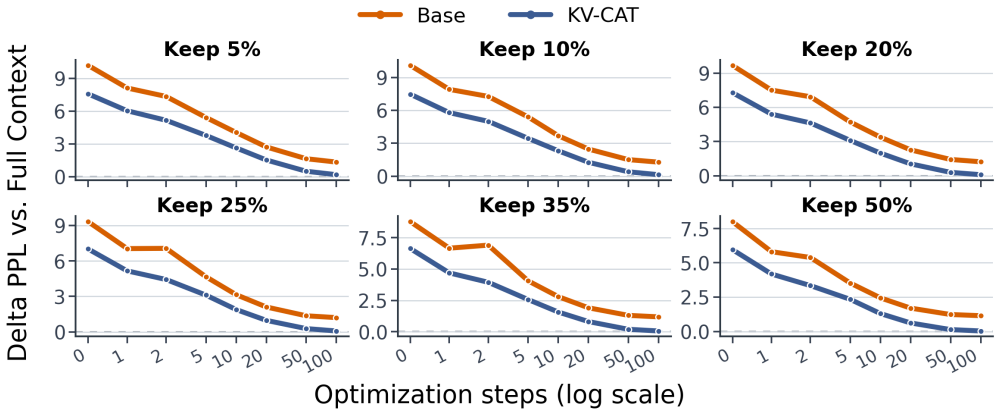


Figure 2: **KV-CAT speeds up gradient-based KV cache compression.** We plot the gap in suffix perplexity under full/compressed-prefix inference throughout gradient-based KV cache optimization. Each panel fixes a different KV keep ratio.

where sg denotes the stop-gradient operation. This term trains the masked forward pass to match the dense model distribution via self-distillation. When the KV-sparsification policy is learnable, we include a budget loss designed to maintain a desirable retention rate. Following Hwang et al. [24], in our implementation, we use:

$$\mathcal{L}_{\text{budget}} = \rho^{-1} FG + (1-\rho)^{-1}(1-F)(1-G), \quad F = \frac{1}{n}\sum_{i=1}^{n} m_i, \quad G = \frac{1}{n}\sum_{i=1}^{n} q_i, \tag{16}$$

where q_i and m_i denote the router's score and binary mask for token i, respectively, and ρ is the target retention rate. The loss is averaged across router layers, and gradients are taken with respect to the scores q_i only. The final term

$$\mathcal{L}_{\text{anchor}} = \frac{1}{n}\sum_{i=1}^{n} -\log p_\theta^{\text{dense}}(a_i \mid a_{<i}), \tag{17}$$

applies standard NTP to the output of the dense forward pass, preserving its distribution and providing a stable teacher for distillation. See Figure 1 for an illustration of this training procedure.

Inference. At evaluation time, we use the unmasked forward pass, on top of which we can apply standard KV cache compression methods. Thus, the output of KV-CAT is a standard transformer whose representations are trained to be more amenable to compression.

5 Empirical Evaluation

We evaluate the effect of KV-CAT on both uncompressed behavior and downstream KV cache compression performance. Our empirical study is guided by four central questions: **(Q1)** Does KV-CAT preserve model performance in the uncompressed setting? **(Q2)** Does KV-CAT make post-hoc KV cache compression more effective at preserving next-token prediction (NTP) quality? **(Q3)** Do these gains translate to improved retrieval from KV-compressed contexts? **(Q4)** Do they transfer to KV-compressed long-context question answering? Across all experiments, we evaluate KV-CAT against **Base**: the same LLM subjected to continued pretraining with standard NTP loss on identical data and training setups. This ensures any performance improvements stem directly from KV-CAT rather than the additional training phase.

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 [47]. 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 maximum learning rate of 10^{-4}. Further details and an evaluation of the computational cost of continued pretraining are provided in Appendix C.1.

KV cache compression methods. We evaluate KV-CAT across two broad families of post-hoc KV cache compression methods. The first consists of lightweight methods that construct a compressed cache directly from the prefix using simple selection or eviction procedures. This family includes

Table 2: **Suffix completion under prefix KV compression.** We report ΔPPL (↓) for a 768-token prefix and 256-token suffix. Each entry reports *Base / KV-CAT*.

	QWEN2.5-0.5B			QWEN2.5-1.5B		
Compression method	10%	20%	40%	10%	20%	40%
Attention Matching	0.479 / **0.422**	0.566 / **0.438**	0.421 / **0.360**	0.664 / **0.320**	1.385 / **0.536**	0.659 / **0.259**
H2O	1.790 / **1.698**	0.825 / **0.775**	0.328 / **0.316**	1.319 / **1.142**	0.618 / **0.495**	0.332 / **0.184**
SnapKV	1.906 / **1.799**	0.919 / **0.833**	0.330 / **0.297**	1.470 / **1.370**	0.705 / **0.562**	0.373 / **0.224**
KVzip	2.746 / **2.389**	1.565 / **1.333**	0.633 / **0.508**	1.976 / **1.654**	1.108 / **0.821**	0.469 / **0.227**
StreamingLLM	2.025 / **1.941**	1.290 / **1.192**	0.502 / **0.471**	1.572 / **1.483**	1.067 / **0.913**	0.512 / **0.393**
KeyDiff	3.986 / **3.426**	2.403 / **2.053**	1.037 / **0.933**	2.846 / **2.543**	1.651 / **1.399**	0.697 / **0.510**

H_2O [65], SnapKV [37], StreamingLLM [57], KVzip [32], and KeyDiff [46]. The second family consists of more expressive optimization-based methods, which fit a compact prefix cache to reproduce the dense model's behavior on a suffix. We consider Attention Matching [67], which fits the compact cache to suffix attention traces, and a gradient-based method adapted from Eyuboglu et al. [16], which directly optimizes the compact KV states to match the dense model's logits on a reconstruction sequence. These two families span a clear cost–quality tradeoff: lightweight methods construct compressed caches using inexpensive, largely fixed procedures, while optimization-based methods solve an example-specific fitting problem and generally achieve higher-quality compression at greater computational cost. For NTP under prefix compression (**Q2**), we evaluate methods from both families to test whether KV-CAT improves NTP under compression across this range of compressors. For retrieval and long-context question answering (QA) (**Q3–Q4**), we focus on the gradient-based method, as the other methods, including Attention Matching, do not preserve sufficient information to achieve meaningful performance even under mild compression. For these experiments, we follow Zweiger et al. [67] and fit the compact cache using 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 to the second repeated `<text>` span. Additional details are provided in Appendix C.

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 [20]. As shown in Table 1, KV-CAT closely matches the base model, with an average drop of 0.02 accuracy points for QWEN2.5-0.5B and an average improvement of 0.29 points for QWEN2.5-1.5B. These results indicate that KV-CAT preserves standard dense behavior rather than trading it off for compressibility. Preservation of dense model behavior is further supported by the uncompressed perplexity evaluation in Appendix D.2, where KV-CAT matches or slightly improves next-token prediction performance across both model sizes.

Table 3: **Needle retrieval from compressed haystack.** Exact-match success rate (%) over five seeds with 200 prompts per seed per keep ratio.

	QWEN2.5-0.5B		QWEN2.5-1.5B	
Keep	Base	KV-CAT	Base	KV-CAT
Full	**99.2**	96.4	87.6	**90.8**
10%	**15.7**	14.5	19.7	**20.4**
20%	**15.6**	14.8	22.7	**25.5**
30%	25.8	**30.5**	42.5	**46.5**
40%	27.6	**38.3**	48.0	**52.6**
50%	33.4	**50.5**	59.4	**65.9**
Mean	23.6	**29.7**	38.5	**42.2**

NTP under KV-compressed prefixes (Q2). We evaluate how well KV-CAT preserves NTP quality after post-hoc compression of the prefix KV cache across compression methods, model sizes, and keep ratios. We sample held-out prefix–suffix pairs from the FineWeb-Edu dataset and evaluate five lightweight eviction methods, which construct the compressed cache from the prefix alone, as well as Attention Matching and gradient-based KV optimization, which use the suffix to fit the compact cache[5]. Thus, the lightweight methods measure prediction quality on an unseen continuation, whereas the optimization-based methods measure how effectively and efficiently a compact cache can be fit under the same optimization procedure. We report ΔPPL between suffix prediction under the full and compressed prefixes over 128 pairs per keep ratio; additional distributional similarity metrics are reported in Appendix D.3. As shown in Table 2, across all methods, keep ratios, and model sizes, KV-CAT improves compression quality, achieving **1.32**× better retention of suffix perplexity on average, with gains reaching **2.58**×. For gradient-based optimization, Figure 2 shows

[5]In **Q3–Q4**, the optimization-based compressor is instead fit using a reconstruction sequence constructed solely from the prefix, thus downstream queries are unseen during compression.

Table 4: **Long-form compressed context QA.** Accuracy (%) on seven LongBench v2 tasks. *Full* corresponds to no compression.

	10% keep		20% keep		50% keep		Full	
Subdomain	Base	KV-CAT	Base	KV-CAT	Base	KV-CAT	Base	KV-CAT
Academic	22.3	**26.6**	22.3	**27.7**	25.5	**26.6**	**25.5**	23.4
Agent history QA	25.0	**30.0**	25.0	**30.0**	**30.0**	**30.0**	**35.0**	30.0
Knowledge graph reasoning	**33.3**	**33.3**	**33.3**	26.7	**46.7**	26.7	33.3	**40.0**
Legal	33.3	**48.5**	27.3	**45.5**	36.4	**42.4**	42.4	**51.5**
Many-shot learning	**23.8**	19.0	**28.6**	**28.6**	23.8	**33.3**	19.0	**28.6**
New language translation	**45.0**	30.0	**35.0**	**35.0**	**35.0**	**35.0**	**35.0**	**35.0**
Table QA	22.2	**27.8**	**22.2**	**22.2**	27.8	**33.3**	**50.0**	38.9
Total	27.1	**30.3**	25.8	**30.8**	29.9	**31.2**	31.7	**32.1**

that KV-CAT achieves better suffix perplexity throughout optimization, requiring up to **5**$\times$ fewer steps to match the base model's performance. For additional details, see Appendix C.3 and for cross-domain evaluation see Appendix D.4.

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 described earlier, training the compacted KVs for 100 steps on the reconstruction sequence. We find that KV-CAT generally improves retrieval accuracy across compression budgets, with larger gains in the moderate-compression regime. As shown in Table 3, across all keep ratios, average accuracy increases by 6.1 points for QWEN2.5-0.5B and 3.7 points for QWEN2.5-1.5B. At 30–50% keep, these gains reach 10.8 and 5.0 points on average, maxing out at 17.1 and 6.5 points, respectively. See additional details in Appendix C.4 and the standard deviation over seeds in Appendix D.6.

Compressed long-context QA (Q4). We evaluate KV-CAT on seven LongBench v2 [3] tasks, compressing each question's context using the reconstruction sequence before prediction. This evaluation extends substantially beyond the 1K-token sequences used for KV-CAT training: for 67.9% of samples, inputs exceed Qwen's context limit and are truncated to fit its full 32K-token window. As shown in Table 4, KV-CAT-trained models surpass base models in average accuracy at every KV retention ratio, with average improvements of up to 19%. These results demonstrate that the downstream gains of KV-CAT persist at these much longer context lengths, even without explicit context-length adaptation. See additional details in Appendix C.5.

6 Related Work

Context and KV cache compression. Prior work reduces long-context inference costs at several levels. Token-level approaches shorten the input through retrieval or prompt compression [35; 26; 27; 36; 44], while learned-memory methods replace long contexts with compact memory or soft representations [6; 10; 42; 58; 9]. More directly related to our setting, post-hoc KV cache compression reduces the cache of pretrained transformers at inference time. Eviction methods retain selected KV slots using attention, recency, or other importance signals [65; 45; 37; 7; 53; 32], while methods such as Cartridges, Attention Matching, and Lexico construct compact latent KV representations [16; 67; 33]. Our work is complementary to these compressors: rather than introducing a new post-hoc compression method, we train the model so that existing compressors can construct higher-quality compact caches.

Training-time sparsification and consistency. Exposing neural networks to sparse or perturbed computation during training has been explored in several other contexts. H-Net reduces the number of tokens processed within a Transformer forward pass through learned token routing [24]. Targeted Dropout trains networks to remain robust to structured parameter removal, facilitating subsequent pruning [17]. Dense-Sparse-Dense training alternates between dense and sparse optimization phases to improve the final dense model [21], while R-Drop encourages consistent predictions across different dropout masks [56]. KV-CAT instead targets internal representations that are inherently more amenable to post-hoc KV cache compression.

7 Conclusion

In this work, we show that KV cache compressibility depends not only on the task or input, but also on the learned representation. Theoretically, we show that almost any sequence-to-vector function admits both highly compressible and inherently non-compressible implementations. Motivated by this, we introduce KV-CAT, a compression-aware training procedure that promotes compressible representations. Through train-time KV sparsification, KV-CAT exposes the model to compression constraints during training, encouraging representations that remain effective with reduced KV caches. Empirically, KV-CAT consistently improves compression performance across tasks, post-hoc compressors, models, and retention budgets. Overall, our results suggest a complementary approach to post-hoc compression methods: training models to become compressible.

Acknowledgements and Disclosures

YG was supported by the UKRI Engineering and Physical Sciences Research Council (EPSRC) CDT in Autonomous and Intelligent Machines and Systems (grant reference EP/S024050/1). MB is partially supported by the EPSRC Turing AI World-Leading Research Fellowship No. EP/X040062/1 and EPSRC AI Hub No. EP/Y028872/1. HM is supported by the Israel Science Foundation through a personal grant (ISF 264/23) and an equipment grant (ISF 532/23). The authors declare no competing interests.

References

[1] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. In *International Conference on Learning Representations*, 2018.

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

[3] 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.

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

[5] 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`.

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

[7] 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.

[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] Aidan N Gomez, Ivan Zhang, Siddhartha Rao Kamalakara, Divyam Madaan, Kevin Swersky, Yarin Gal, and Geoffrey E Hinton. Learning sparse networks using targeted dropout. *arXiv preprint arXiv:1905.13678*, 2019.

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

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

[20] 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.

[21] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. DSD: Dense-sparse-dense training for deep neural networks. *arXiv preprint arXiv:1607.04381*, 2016.

[22] Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. Relu deep neural networks and linear finite elements. *Journal of Computational Mathematics*, 38(3):502–527, 2020.

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

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

[25] 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.

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

[27] 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.

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

[29] Samuel Karlin and William J. Studden. *Tchebycheff Systems: With Applications in Analysis and Statistics*. Interscience Publishers, New York, 1966.

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

[31] 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.

[32] 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.

[33] 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.

[34] Alexander Kozachinskiy, Felipe Urrutia, Hector Orellana, Tomasz Steifer, Germán Pizarro, Matías Fuentes, Francisco Meza Vásquez, Cristian Buc Calderon, and Cristóbal Rojas. Strassen attention, split vc dimension and compositionality in transformers. *Advances in Neural Information Processing Systems*, 38:9665–9696, 2026.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

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

[40] 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.

[41] 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/`.

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

[43] 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.

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

[45] 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.

[46] Junyoung Park, Dalton Jones, Matthew Morse, Raghavv Goel, Mingu Lee, and Christopher Lott. KeyDiff: Key similarity-based KV cache eviction for long-context LLM inference in resource-constrained environments. *Advances in Neural Information Processing Systems*, 38: 5983–6019, 2026.

[47] 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.

[48] Binghui Peng, Srini Narayanan, and Christos Papadimitriou. On limitations of the transformer architecture. *arXiv preprint arXiv:2402.08164*, 2024.

[49] 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`.

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

[51] 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`.

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

[53] 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.

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

[55] Peihao Wang, Shenghao Yang, Shu Li, Zhangyang Wang, and Pan Li. Polynomial width is sufficient for set representation with high-dimensional features. In *International Conference on Learning Representations*, volume 2024, pages 52223–52252, 2024.

[56] Lijun Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, Tie-Yan Liu, et al. R-Drop: Regularized dropout for neural networks. *Advances in neural information processing systems*, 34:10890–10905, 2021.

[57] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pages 21875–21895, 2024.

[58] 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`.

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

[60] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in neural information processing systems*, 37:115491–115522, 2024.

[61] 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.

[62] 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.

[63] 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.

[64] 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/`.

[65] 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.

[66] 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.

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

A Theory

A.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

$$[\boldsymbol{X}, \boldsymbol{Y}] \in \mathbb{R}^{n \times (d+d')}. \tag{18}$$

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

$$\begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{Z} \end{bmatrix} \in \mathbb{R}^{(n+n') \times d}. \tag{19}$$

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

$$[\boldsymbol{v}, \boldsymbol{u}] \in \mathbb{R}^{n \times 2}. \tag{20}$$

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

$$\text{AttnHead} : (\mathbb{R}^{d_{\text{in}}})^* \to (\mathbb{R}^{d_{\text{out}}})^* \tag{21}$$

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

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V, \tag{22}$$

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

$$\text{AttnHead}(\boldsymbol{X}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}} + \boldsymbol{M}\right)\boldsymbol{V}, \tag{23}$$

where the softmax is applied row-wise and $\boldsymbol{M} \in \mathbb{R}^{n \times n}$ is the causal mask, with $\boldsymbol{M}_{ij} = 0$ if $j \le i$ and $\boldsymbol{M}_{ij} = -\infty$ otherwise.

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

$$\text{Block} : (\mathbb{R}^{d_{\text{in}}})^* \to (\mathbb{R}^{d_{\text{out}}})^* \tag{24}$$

defined as follows. For a sequence

$$\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^\top \in \mathbb{R}^{n \times d_{\text{in}}}, \tag{25}$$

we define

$$\text{Block}(\boldsymbol{X}) = \text{FFN}([\text{head}_1(\boldsymbol{X}), \ldots, \text{head}_h(\boldsymbol{X})]\boldsymbol{W}_O + \boldsymbol{X}), \tag{26}$$

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

$$\text{FFN}(\boldsymbol{X})_i = \text{FFN}(\boldsymbol{x}_i), \quad i = 1, \ldots, n, \tag{27}$$

with

$$\text{FFN}(\boldsymbol{x}) = \sigma(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2, \tag{28}$$

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

Note. In the theoretical analysis, we omit layer normalization, and the residual connection following the FFN for simplicity, as is standard in prior theoretical treatments (See e.g. [52; 61]). Our results extend to the full architecture with only minor modifications. All empirical evaluations are conducted with these components included.

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

$$\text{emb} : A \times \mathbb{N} \to \mathbb{R}^{d_{\text{model}}}. \tag{29}$$

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

$$\text{emb}(\boldsymbol{a}) = [\text{emb}(a_1, 1), \ldots, \text{emb}(a_n, n)]^\top \in \mathbb{R}^{n \times d_{\text{model}}}. \tag{30}$$

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

$$\mathsf{M} = (\text{emb}, \text{Block}_1, \ldots, \text{Block}_L), \tag{31}$$

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

The transformer M defines a function

$$\mathsf{M} : A^* \to \mathbb{R}^{d_L} \tag{32}$$

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

$$\boldsymbol{X}^{(0)} = \text{emb}(\boldsymbol{a}), \qquad \boldsymbol{X}^{(\ell)} = \text{Block}_\ell(\boldsymbol{X}^{(\ell-1)}), \quad \ell = 1, \ldots, L. \tag{33}$$

Then

$$\mathsf{M}(\boldsymbol{a}) = \boldsymbol{X}_n^{(L)}, \tag{34}$$

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

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

$$\mathsf{c}_\ell : (\mathbb{R}^{d_\ell} \times \mathbb{R}^{d_\ell})^* \to (\mathbb{R}^{d_\ell} \times \mathbb{R}^{d_\ell})^* \tag{35}$$

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

Concretely, for

$$(\boldsymbol{K}, \boldsymbol{V}) = \big([\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n]^\top, [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]^\top\big), \tag{36}$$

we write

$$\mathsf{c}_\ell(\boldsymbol{K}, \boldsymbol{V}) = (\tilde{\boldsymbol{K}}, \tilde{\boldsymbol{V}}) = \big([\tilde{\boldsymbol{k}}_1, \ldots, \tilde{\boldsymbol{k}}_{r(n)}]^\top, [\tilde{\boldsymbol{v}}_1, \ldots, \tilde{\boldsymbol{v}}_{r(n)}]^\top\big), \tag{37}$$

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

Given a transformer $\mathsf{M} = (\text{emb}, \text{Block}_1, \ldots, \text{Block}_L)$, 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

$$\boldsymbol{Q}_{\boldsymbol{b}} = \boldsymbol{Y}\boldsymbol{W}_Q, \quad \boldsymbol{K}_{\boldsymbol{b}} = \boldsymbol{Y}\boldsymbol{W}_K, \quad \boldsymbol{V}_{\boldsymbol{b}} = \boldsymbol{Y}\boldsymbol{W}_V. \tag{38}$$

We then compress the context KV pairs:

$$(\tilde{\boldsymbol{K}}_{\boldsymbol{a}}, \tilde{\boldsymbol{V}}_{\boldsymbol{a}}) = \mathsf{c}_\ell(\boldsymbol{K}_{\boldsymbol{a}}, \boldsymbol{V}_{\boldsymbol{a}}). \tag{39}$$

The resulting attention computation is

$$\mathrm{AttnHead}(\boldsymbol{Y}) = \mathrm{softmax}\left(\frac{1}{\sqrt{d_k}}\boldsymbol{Q}_{\boldsymbol{b}} \begin{bmatrix}\tilde{\boldsymbol{K}}_{\boldsymbol{a}} \\ \boldsymbol{K}_{\boldsymbol{b}}\end{bmatrix}^\top\right) \begin{bmatrix}\tilde{\boldsymbol{V}}_{\boldsymbol{a}} \\ \boldsymbol{V}_{\boldsymbol{b}}\end{bmatrix}. \tag{40}$$

Definition A.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

$$\|\mathsf{M}([\boldsymbol{a}, \boldsymbol{b}]) - \mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})\| < \varepsilon. \tag{41}$$

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. [52]; Yehudai et al. [61] for examples.

A.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

$$\mathrm{emb}(j, i) = \boldsymbol{u}_j + \boldsymbol{p}_i, \tag{42}$$

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

$$\mathrm{Block}_1(\boldsymbol{X})_i \approx \rho_1(\boldsymbol{X}_i) \tag{43}$$

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

$$\rho_1(\boldsymbol{u}_j + \boldsymbol{p}_i) = \begin{bmatrix}\boldsymbol{e}_j \\ \boldsymbol{0}\end{bmatrix}. \tag{44}$$

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; 23]).

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

$$\mathrm{Block}_2(\boldsymbol{X})_i = \frac{1}{i}\sum_{j=1}^{i} \boldsymbol{x}_j \tag{45}$$

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

$$\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{0}, \tag{46}$$

so that all attention weights are uniform, and choosing

$$\boldsymbol{W}_V = \boldsymbol{I}, \quad \boldsymbol{W}_O = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}. \tag{47}$$

This choice extracts averages the vectors and then extracts the first half it in the second half. After applying a residual connection, the intermediate representation of the i-th token is then

$$\begin{bmatrix} \boldsymbol{x}_i \\ \frac{1}{i}\sum_{j=1}^{i} \boldsymbol{x}_j \end{bmatrix} \tag{48}$$

Finally, the feedforward network is taken to be

$$\mathrm{FFN}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{y}, \tag{49}$$

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

$$\mathsf{M}(\boldsymbol{a}) \approx \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{e}_{a_i} = \left(\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{a_i=1},\ \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{a_i=2},\ \ldots,\ \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{a_i=m} \right) = f_{\mathrm{hist}}(\boldsymbol{a}), \tag{50}$$

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 during 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

$$\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) = \frac{1}{r(n)+k}\left(\sum_{i=1}^{r(n)} \tilde{\boldsymbol{v}}_i + \sum_{i=1}^{k} \boldsymbol{e}_{b_i} \right). \tag{51}$$

$$\mathsf{M}([\boldsymbol{a}, \boldsymbol{b}]) = \frac{1}{n+k}\left(\sum_{i=1}^{n} \boldsymbol{e}_{a_i} + \sum_{i=1}^{k} \boldsymbol{e}_{b_i} \right). \tag{52}$$

For notational convenience, define

$$\tilde{\boldsymbol{v}} = \frac{1}{r(n)+k}\sum_{i=1}^{r(n)} \tilde{\boldsymbol{v}}_i, \quad \tilde{\boldsymbol{a}} = \frac{1}{n+k}\sum_{i=1}^{n} \boldsymbol{e}_{a_i}, \quad \tilde{\boldsymbol{b}} = \sum_{i=1}^{k} \boldsymbol{e}_{b_i}. \tag{53}$$

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

$$\|\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a}, \boldsymbol{b}])\| = \left\| \tilde{\boldsymbol{v}} - \tilde{\boldsymbol{a}} + \left(\frac{1}{r(n)+k} - \frac{1}{n+k} \right) \tilde{\boldsymbol{b}} \right\|. \tag{54}$$

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

$$\|\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a}, \boldsymbol{b}])\| \geq \left| \tilde{\boldsymbol{v}}_1 - \tilde{\boldsymbol{a}}_1 + \frac{k(n - r(n))}{(n+k)(r(n)+k)} \right|. \tag{55}$$

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

$$\|\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a}, \boldsymbol{b}])\| \geq |\tilde{\boldsymbol{v}}_1 - \tilde{\boldsymbol{a}}_1|. \tag{56}$$

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

$$\|\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a}, \boldsymbol{b}])\| \geq \frac{k(n - r(n))}{2(n + k)(r(n) + k)}. \tag{57}$$

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

$$\|\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) - \mathsf{M}([\boldsymbol{a}, \boldsymbol{b}])\| \geq \frac{N - 1 - r(N - 1)}{2(N)(r(N - 1) + 1)} = C > 0, \tag{58}$$

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

$$\mathrm{emb}(j, i) = \boldsymbol{u}_j + \boldsymbol{p}_i, \tag{59}$$

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 $[m] \times [N]$.

Additionally, for $m \geq 2$, choose the positional vectors $\boldsymbol{p}_i$ to be distinct unit vectors, with token vectors chosen to preserve injectivity of emb. For $m = 1$, the histogram is constant and the claim is immediate.

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

$$\mathrm{Block}_1(\boldsymbol{X})_i \approx \rho_1(\boldsymbol{X}_i) \tag{60}$$

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

$$\rho_1(\boldsymbol{u}_j + \boldsymbol{p}_i) = \begin{bmatrix} \boldsymbol{e}_j \\ \boldsymbol{p}_i \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}. \tag{61}$$

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; 23].

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

$$\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{0}, \boldsymbol{W}_O = \boldsymbol{I} \tag{62}$$

so that all attention weights are uniform, and choosing

$$\boldsymbol{W}_V = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}. \tag{63}$$

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

$$\begin{bmatrix} \boldsymbol{e}_{a_i} \\ \boldsymbol{p}_i \\ \frac{1}{i}\sum_{j=1}^{i} \boldsymbol{e}_{a_j} \\ \mathbf{0} \end{bmatrix}. \tag{64}$$

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

$$\rho_2(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \begin{cases} \boldsymbol{y} & \text{if } \boldsymbol{z} = 0 \\ \frac{j-i+1}{j}\boldsymbol{y} & \text{if } \boldsymbol{z} = \frac{\boldsymbol{p}_i}{k+1} \text{ and } \boldsymbol{x} = \boldsymbol{p}_j. \end{cases} \tag{65}$$

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 64 and 65 imply that, in the uncompressed setting, for any sequence $\boldsymbol{a}$ of length $n \leq N$, we recover as before

$$\mathsf{M}(\boldsymbol{a}) \approx \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{e}_{a_i} = \left(\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{a_i=1},\ \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{a_i=2},\ \ldots,\ \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{a_i=m} \right) = f_{\text{hist}}(\boldsymbol{a}), \tag{66}$$

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 $\boldsymbol{W}_O = \mathbf{0}$ in the first block, c_1 does not affect the forward pass of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}$.

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

$$\mathsf{c}_2(\boldsymbol{K}_{\boldsymbol{a}}, \boldsymbol{V}_{\boldsymbol{a}}) = \Big(\mathbf{0}, \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \sum_{i=1}^{n} \boldsymbol{e}_{a_i} \\ \boldsymbol{p}_n \end{bmatrix}\Big). \tag{67}$$

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

$$\begin{bmatrix} \boldsymbol{e}_{b_k} \\ \boldsymbol{p}_{n+k} \\ \frac{1}{k+1}\Big(\sum_{i=1}^{n} \boldsymbol{e}_{a_i} + \sum_{i=1}^{k} \boldsymbol{e}_{b_i}\Big) \\ \frac{1}{k+1}\boldsymbol{p}_n \end{bmatrix}. \tag{68}$$

Applying ρ_2 and using equation 65, we obtain

$$\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b}) \approx \frac{1 + (n+k) - n}{n+k} \cdot \frac{1}{k+1}\Big(\sum_{i=1}^{n} \boldsymbol{e}_{a_i} + \sum_{i=1}^{k} \boldsymbol{e}_{b_i}\Big) = \frac{1}{n+k}\sum_{i=1}^{n+k} \boldsymbol{e}_{c_i}, \tag{69}$$

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

□

A.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 A.7) establishes the existence of compressible transformer approximations, and the second (Lemma A.8) demonstrates the existence of non-compressible ones. The theorem then follows immediately.

Remark (Matching architectures). For clarity, the constructions in Lemma A.7 and Lemma A.8 may use different intermediate widths and depths. These constructions can be implemented with the same architecture by padding their hidden representations with zero coordinates to a common width, enlarging their feedforward networks with unused units, and inserting identity blocks into the shallower construction. The projections and decoders are extended to ignore the padded coordinates. Identity blocks have zero attention output and an identity feedforward map, so compression of their KV caches has no effect. These modifications preserve the functions computed and the respective compressibility properties.

Lemma A.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 $\boldsymbol{a} = (a_1, \dots, a_n) \in A^n$ with $n \le N$,*
$$\|f(\boldsymbol{a}) - \mathsf{M}(\boldsymbol{a})\| < \varepsilon. \tag{70}$$
2. *(**Maximal compressibility**) There exists a KV cache compression policy C with compression budget*
$$r(n) \equiv 1 \tag{71}$$
such that for every prefix $\boldsymbol{a} \in A^n$ and suffix $\boldsymbol{b} \in A^k$ with $k + n \le N$,
$$\|\mathsf{M}([\boldsymbol{a}, \boldsymbol{b}]) - \mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})\| < \varepsilon. \tag{72}$$

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

$$\mathrm{emb}(a, i) = \boldsymbol{u}_a + \boldsymbol{p}_i \tag{73}$$

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

$$(a, i) \ne (b, j) \;\Rightarrow\; \mathrm{emb}(a, i) \ne \mathrm{emb}(b, j). \tag{74}$$

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 $\boldsymbol{u}_\emptyset \ne \mathrm{emb}(a, i)$ for all $a \in A$ and $i \in \mathbb{N}$. For a sequence $\boldsymbol{a} = (a_1, \dots, a_n) \in A^*$ with $n \le N$, define

$$U_{\boldsymbol{a}} = \left\{ \boldsymbol{P}[\mathrm{emb}(a_1, 1), \dots, \mathrm{emb}(a_n, n), \boldsymbol{u}_\emptyset, \dots, \boldsymbol{u}_\emptyset]^\top \mid \boldsymbol{P} \in \mathbb{R}^{N \times N} \text{ permutation matrix} \right\} \subset \mathbb{R}^{N \times d_0}. \tag{75}$$

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 $\{U_{\boldsymbol{a}} : \boldsymbol{a} \in A^*, |\boldsymbol{a}| \le N\}$ are pairwise disjoint. Consequently, there exists a continuous function

$$\bar{f} : \mathbb{R}^{N \times d_0} \to \mathbb{R}^{d_{\mathrm{out}}} \tag{76}$$

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

$$\bar{f}(\boldsymbol{v}_1, \dots, \boldsymbol{v}_N) = f(\boldsymbol{a}). \tag{77}$$

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

$$\bar{f}(\boldsymbol{v}_1, \dots, \boldsymbol{v}_N) = \bar{f}(\boldsymbol{v}_{\sigma(1)}, \dots, \boldsymbol{v}_{\sigma(N)}). \tag{78}$$

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

$$\tilde{f}(\boldsymbol{v}_1, \dots, \boldsymbol{v}_N) = \frac{1}{|S_N|} \sum_{\sigma \in S_N} \bar{f}(\boldsymbol{v}_{\sigma(1)}, \dots, \boldsymbol{v}_{\sigma(N)}). \tag{79}$$

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

$$\tilde{f}(\boldsymbol{V}) = \bar{f}(\boldsymbol{V}), \tag{80}$$

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

Because $\bar{f}$ is continuous and permutation-invariant, it follows from Zaheer et al. [62]; Wang et al. [55] that there exists a pair of functions [6] $\phi : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ and $\rho : \mathbb{R}^{d_1} \to \mathbb{R}^{d_{\text{out}}}$ such that

$$\bar{f}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N) = \rho\left(\sum_{i=1}^{N} \phi(\boldsymbol{v}_i)\right). \tag{81}$$

We further assume without loss of generality that $\phi(\boldsymbol{u}_\emptyset) = 0$. We now construct a compressible transformer $\mathsf{M} = (\mathrm{emb}, \mathrm{Block}_1, \mathrm{Block}_2)$ approximating f using this decomposition. The construction of this transformer closely 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,

$$\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{W}_V = \boldsymbol{W}_O = \boldsymbol{0}, \tag{82}$$

so that the block reduces to its feedforward component.

For the rest of the proof, assume for simplicity that $d_0 = d_1$ (this can be easily achieved using padding). Let $\tilde{\phi} : \mathbb{R}^{d_0} \to \mathbb{R}^{4d_0}$ be a continuous function satisfying

$$\tilde{\phi}(\mathrm{emb}(a, i)) = \begin{bmatrix} \phi(\mathrm{emb}(a, i)) \\ \boldsymbol{p}_i \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}. \tag{83}$$

for all $a \in A$ and $i \in [N]$ where all 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 feedforward networks are universal approximators of continuous functions on compact sets [14; 23]). Thus, for any sequence $\boldsymbol{a} = (a_1, \ldots, a_n)$,

$$\mathrm{Block}_1(\mathrm{emb}(\boldsymbol{a})) \approx \begin{bmatrix} \tilde{\phi}(\mathrm{emb}(a_1, 1)) \\ \vdots \\ \tilde{\phi}(\mathrm{emb}(a_n, n)) \end{bmatrix}. \tag{84}$$

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

$$\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{0}, \boldsymbol{W}_O = \boldsymbol{I} \tag{85}$$

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 last d_1 coordinates the same, and zero out the remaining coordinates. That is:

$$\boldsymbol{W}_V = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}. \tag{86}$$

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

$$\begin{bmatrix} \phi(\mathrm{emb}(a_i, i)) \\ \boldsymbol{p}_i \\ \frac{1}{i}\sum_{j=1}^{i} \phi(\mathrm{emb}(a_j, j)) \\ \boldsymbol{0} \end{bmatrix}. \tag{87}$$

[6] Note that some functions require $d_1 = O(N)$. This dependence on N reflects the generality of the result: approximating arbitrary sequence functions may itself require transformer dimension growing with N [52; 61; 48; 34]. It is not intrinsic to the compressibility separation; for example, the histogram constructions in Appendix A.2 establish the same phenomenon with hidden dimension independent of N.

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

$$\begin{aligned} \tilde{\rho}(\boldsymbol{w}, \boldsymbol{p}_j, \boldsymbol{y}, \mathbf{0}) &= \rho(j\boldsymbol{y}), && 1 \le j \le N, \\ \tilde{\rho}\left(\boldsymbol{w}, \boldsymbol{p}_j, \boldsymbol{y}, \frac{\boldsymbol{p}_q}{j-q+1}\right) &= \rho((j-q+1)\boldsymbol{y}), && 1 \le q < j \le N. \end{aligned} \tag{88}$$

such a continuous function exists as for each fixed j, the nonzero vectors $\boldsymbol{p}_q/(j-q+1)$ have distinct norms as q varies, and none equals zero.

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

$$\rho\left(n \cdot \frac{1}{n}\sum_{j=1}^{n} \phi(\mathrm{emb}(a_j, j))\right) = \rho\left(\sum_{j=1}^{n} \phi(\mathrm{emb}(a_j, j))\right) = f(\boldsymbol{a}). \tag{89}$$

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 72.

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

$$\mathsf{c}_1(\boldsymbol{K}, \boldsymbol{V}) = (\mathbf{0}, \mathbf{0}), \tag{90}$$

and

$$\mathsf{c}_2(\boldsymbol{K}, \boldsymbol{V}) = (\tilde{\boldsymbol{K}}, \tilde{\boldsymbol{V}}), \tag{91}$$

with $\tilde{\boldsymbol{K}}, \tilde{\boldsymbol{V}}$ defined as followed: Let $P_3 : \mathbb{R}^{4d_1} \to \mathbb{R}^{d_1}$ extract the third vector block. For an input cache with $n \ge 1$ rows, set

$$\tilde{\boldsymbol{K}} = \mathbf{0}, \qquad \tilde{\boldsymbol{V}} = \left[\mathbf{0}, \mathbf{0}, \sum_{j=1}^{n} P_3(\boldsymbol{v}_j), \boldsymbol{p}_n\right]. \tag{92}$$

The vector $\boldsymbol{p}_n$ is obtained from the fixed positional table using the number n of input cache rows. Thus the policy uses only its KV inputs and their length. On ideal cached values, the third block equals $\sum_{j=1}^{n} \phi(\mathrm{emb}(a_j, j))$.

Clearly, $r(n) = 1$. Additionally, 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 approximate

$$\begin{bmatrix} \tilde{\phi}(\mathrm{emb}(b_1, 1+n)) \\ \vdots \\ \tilde{\phi}(\mathrm{emb}(b_k, k+n)) \end{bmatrix}. \tag{93}$$

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 suffix position $r \in [k]$ in the second block. Since $\boldsymbol{W}_Q = 0$, causal attention is uniform over the one compressed prefix row and the first r suffix rows, with weight $1/(r+1)$. By the definition of c_2, the state after attention and the residual connection is approximately

$$\begin{bmatrix} \phi(\mathrm{emb}(b_r, n+r)) \\ \boldsymbol{p}_{n+r} \\ \dfrac{1}{r+1}\left(\sum_{j=1}^{n} \phi(\mathrm{emb}(a_j, j)) + \sum_{t=1}^{r} \phi(\mathrm{emb}(b_t, n+t))\right) \\ \dfrac{\boldsymbol{p}_n}{r+1} \end{bmatrix}. \tag{94}$$

Finally, after applying the feed forward network FFN , by choosing $\varepsilon', \varepsilon''$ small enough, Equation 88 implies that thefinal output of $\mathsf{M}_{\mathsf{C},\boldsymbol{a}}(\boldsymbol{b})$, corresponding to $r = k$ is approximately

$$\rho\left((k+n-n+1)\cdot\frac{1}{k+1}\big(\sum_{j=1}^{n}\phi(\mathrm{emb}(a_j,j))+\sum_{j=1}^{k}\phi(\mathrm{emb}(b_j,n+j))\big)\right)\approx f([\boldsymbol{a},\boldsymbol{b}]) \quad (95)$$

Choosing the approximation tolerances sufficiently small ensures that both the full and compressed outputs are within $\varepsilon/2$ of $f([\boldsymbol{a}, \boldsymbol{b}])$. By the triangle inequality, their distance is less than ε, completing the proof.

□

Lemma A.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 exist two suffix sequences $\boldsymbol{b}^1$, $\boldsymbol{b}^2$ both of length $k \le N - n$ such that*

$$f([\bar{\boldsymbol{a}},\boldsymbol{b}^1]) \neq f([\bar{\boldsymbol{a}},\boldsymbol{b}^2]) \quad (96)$$

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 $n \le N$,*
$$\|f(\boldsymbol{a}) - \mathsf{M}(\boldsymbol{a})\| < \varepsilon. \quad (97)$$
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
$$\|\mathsf{M}([\bar{\boldsymbol{a}},\boldsymbol{b}]) - \mathsf{M}_{\mathsf{C},\bar{\boldsymbol{a}}}(\boldsymbol{b})\| > C. \quad (98)$$

Proof. By the same argument as in the proof of Lemma A.7, there exist continuous functions

$$\phi_0 : \mathbb{R}^{d_0} \longrightarrow \mathbb{R}^{d_1-1}, \qquad \rho_0 : \mathbb{R}^{d_1-1} \longrightarrow \mathbb{R}^{d_{\mathrm{out}}} \quad (99)$$

such that, for every sequence $\boldsymbol{a} = (a_1, \ldots, a_m)$ of length $m \le N$,

$$f(\boldsymbol{a}) = \rho_0\left(\sum_{i=1}^{m}\phi_0(\mathrm{emb}(a_i,i))\right). \quad (100)$$

Let $\bar{\boldsymbol{a}}$ be the prefix from the lemma statement, of length n, and let $\boldsymbol{b}^1, \boldsymbol{b}^2 \in A^k$ satisfy

$$f([\bar{\boldsymbol{a}},\boldsymbol{b}^1]) \neq f([\bar{\boldsymbol{a}},\boldsymbol{b}^2]). \quad (101)$$

Choose $j_* \in [k]$ with $b^1_{j_*} \neq b^2_{j_*}$, and put $s_* = n + j_*$ and $a_* = b^1_{j_*}$. Since the token-position embeddings are distinct and form a finite set, there is a continuous scalar function q satisfying

$$q(\mathrm{emb}(a,i)) = \mathbf{1}\{i = s_* \text{ and } a = a_*\}, \qquad a \in A,\ i \in [N]. \quad (102)$$

Replace ϕ_0 and ρ_0 by

$$\phi(\boldsymbol{x}) = (\phi_0(\boldsymbol{x}), q(\boldsymbol{x})), \qquad \rho(\boldsymbol{z}, t) = \rho_0(\boldsymbol{z}). \quad (103)$$

The sum decomposition is unchanged:

$$f(\boldsymbol{a}) = \rho\left(\sum_{i=1}^{m}\phi(\mathrm{emb}(a_i,i))\right). \quad (104)$$

Define the convex set using all valid sequences:

$$S(\boldsymbol{a}) := \sum_{i=1}^{|\boldsymbol{a}|} \phi(\mathrm{emb}(a_i, i)), \qquad H := \mathrm{conv}\{S(\boldsymbol{a}) : \boldsymbol{a} \in A^m,\ 1 \le m \le N\}. \tag{105}$$

Let $\pi : \mathbb{R}^{d_1} \to \mathbb{R}$ extract the appended coordinate. Every valid sequence has $\pi(S(\boldsymbol{a})) \in \{0, 1\}$. Moreover,

$$\pi(S([\bar{\boldsymbol{a}}, \boldsymbol{b}^1])) = 1, \qquad \pi(S([\bar{\boldsymbol{a}}, \boldsymbol{b}^2])) = 0. \tag{106}$$

Consequently, $\pi(H) = [0, 1]$. Set $\eta = \varepsilon/2$. Since H is compact and ρ is continuous, choose $\boldsymbol{u}_0 \in \mathbb{R}^{d_{\mathrm{out}}}$ such that

$$\mathrm{dist}(\boldsymbol{u}_0, \rho(H)) > C + \eta. \tag{107}$$

We construct the transformer similarly to the one in Lemma A.7. Its first block has zero attention output and produces tokenwise representations with four vector blocks,

$$\tilde{\phi}(\mathrm{emb}(a, i)) = \begin{bmatrix} \phi(\mathrm{emb}(a, i)) \\ \boldsymbol{p}_i \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}. \tag{108}$$

where the $\boldsymbol{p}_i$ are distinct positional vectors and dimensions are padded as necessary. Choose the first feedforward map to realize these values exactly on the finite embedding set. In the second block, set $\boldsymbol{W}_Q = \boldsymbol{W}_K = 0$. The value projection copies the first vector block into the third, and take

$$\boldsymbol{W}_O = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \boldsymbol{I} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{109}$$

In particular, only the third vector block receives an attention update, even when the compressed values are arbitrary. Define

$$\delta := \frac{1}{2}\left(\frac{n+k}{n-1+k} - 1\right) = \frac{1}{2(n+k-1)}, \qquad H_t := \{\boldsymbol{z} \in \mathbb{R}^{d_1} : \mathrm{dist}(\boldsymbol{z}, H) < t\}. \tag{110}$$

We construct the second feedforward function using Lemma A.9 [7] Choosing $\epsilon' > 0$ sufficiently small as required by that lemma, It provides a finite ReLU MLP $\bar{\rho}$ such that, for every $i \in [N]$,

$$\|\bar{\rho}(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) - \rho(i\boldsymbol{y})\| < \eta \quad \text{if} \quad \|\boldsymbol{x} - \boldsymbol{p}_i\| < \epsilon' \text{ and } i\boldsymbol{y} \in H_{\delta/2}, \tag{111}$$

whereas

$$\bar{\rho}(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \boldsymbol{u}_0 \quad \text{if} \quad \|\boldsymbol{x} - \boldsymbol{p}_i\| < \epsilon' \text{ and } i\boldsymbol{y} \notin H_{\delta}. \tag{112}$$

The behavior in the transition region is immaterial. For any uncompressed sequence $\boldsymbol{a}$ of length $m \le N$, the input to this decoder at its final token is exactly

[7] Lemma A.9 uses MLPs of depth greater than two, whereas our transformer definition follows the standard convention of using a single two-layer MLP in each block. The lemma can nevertheless be applied directly by using additional transformer blocks and setting the attention output matrix $\boldsymbol{W}_O$ in each added block to zero, so that the resulting sequence of blocks is equivalent to a deeper MLP.

$$\left(\phi(\mathrm{emb}(a_m, m)), \boldsymbol{p}_m, \frac{S(\boldsymbol{a})}{m}, \mathbf{0}\right). \tag{113}$$

Since $S(\boldsymbol{a}) \in H \subset H_{\delta/2}$, the first decoder condition gives

$$\|\mathsf{M}(\boldsymbol{a}) - f(\boldsymbol{a})\| = \|\mathsf{M}(\boldsymbol{a}) - \rho(S(\boldsymbol{a}))\| < \eta < \varepsilon. \tag{114}$$

And so Equation 97 holds.

Now let $\mathsf{C} = (\mathsf{c}_1, \ldots, \mathsf{c}_L)$ be any compression policy with $r(n) < n$. The first block have zero attention output, so cache compression cannot affect its output. Let $\widetilde{\boldsymbol{v}} \in \mathbb{R}^{d_1}$ be the sum of the third vector blocks of the $r(n)$ compressed values at the second block. This vector is arbitrary, but depends only on the fixed prefix and the policy, not on the suffix.

At the final token of a suffix $\boldsymbol{b} \in A^k$, the second attention block has uniform weights $1/(r(n)+k)$. Therefore

$$\mathsf{M}_{\mathsf{C},\bar{\boldsymbol{a}}}(\boldsymbol{b}) = \bar{\rho}\left(\phi(\mathrm{emb}(b_k, n+k)), \boldsymbol{p}_{n+k}, \frac{1}{r(n)+k}\left(\widetilde{\boldsymbol{v}} + \sum_{j=1}^{k} \phi(\mathrm{emb}(b_j, n+j))\right), \mathbf{0}\right). \tag{115}$$

Set

$$\alpha := \frac{n+k}{r(n)+k} > 1, \qquad Z(\boldsymbol{b}) := \alpha\left(\widetilde{\boldsymbol{v}} + \sum_{j=1}^{k} \phi(\mathrm{emb}(b_j, n+j))\right), \qquad \beta := \alpha\pi(\widetilde{\boldsymbol{v}}). \tag{116}$$

The two fixed witness suffixes satisfy

$$\pi(Z(\boldsymbol{b}^1)) = \beta + \alpha, \qquad \pi(Z(\boldsymbol{b}^2)) = \beta. \tag{117}$$

This means that there exists a choice $\bar{\boldsymbol{b}} \in \{\boldsymbol{b}^1, \boldsymbol{b}^2\}$ with

$$\mathrm{dist}(Z(\bar{\boldsymbol{b}}), H) \geq \mathrm{dist}(\pi(Z(\bar{\boldsymbol{b}})), [0,1]) \geq \frac{\alpha - 1}{2} = \delta, \tag{118}$$

where the first inequallity follows since coordinate projection is 1-Lipschitz and $\pi(H) = [0,1]$.

Therefore Equation 112, with $i = n + k$, gives

$$\mathsf{M}_{\mathsf{C},\bar{\boldsymbol{a}}}(\bar{\boldsymbol{b}}) = \boldsymbol{u}_0. \tag{119}$$

On the other hand, $S([\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}]) \in H$, and the uncompressed output is within η of $\rho(S([\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}]))$. Thus

$$\begin{aligned} &\|\mathsf{M}([\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}]) - \mathsf{M}_{\mathsf{C},\bar{\boldsymbol{a}}}(\bar{\boldsymbol{b}})\| \\ &\quad \geq \|\rho(S([\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}])) - \boldsymbol{u}_0\| - \|\mathsf{M}([\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}]) - \rho(S([\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}]))\| \\ &\quad > C + \eta - \eta = C. \end{aligned} \tag{120}$$

This completes the proof.

□

Note. We believe that a finer-grained analysis of transformer compressibility may be possible through the theory of Chebyshev systems [29], a classical framework with broad applications across both pure and applied mathematics [28; 40; 15; 30]. 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.

Lemma A.9 (Localized ReLU approximation)**.** *Let $H \subset \mathbb{R}^d$ be compact, let $\rho : \mathbb{R}^d \to \mathbb{R}^{d_{\text{out}}}$ be continuous, let $\boldsymbol{p}_1, \dots, \boldsymbol{p}_N$ be distinct vectors, and let $\boldsymbol{u}_0 \in \mathbb{R}^{d_{\text{out}}}$. Then, for every $\delta > 0$, there exists $\epsilon_0 > 0$ such that, for every $0 < \epsilon' < \epsilon_0$ and every $\epsilon'' > 0$, there exists a finite ReLU MLP $\bar{\rho}$ satisfying, for every $i \in [N]$,*

$$|\bar{\rho}(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) - \rho(i \cdot \boldsymbol{y})| < \epsilon'' \tag{121}$$

whenever

$$\|\boldsymbol{x} - \boldsymbol{p}_i\| < \epsilon' \qquad \textit{and} \qquad i \cdot \boldsymbol{y} \in H_{\delta/2},$$

and

$$\bar{\rho}(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \boldsymbol{u}_0 \tag{122}$$

whenever

$$\|\boldsymbol{x} - \boldsymbol{p}_i\| < \epsilon' \qquad \textit{and} \qquad i \cdot \boldsymbol{y} \notin H_{\delta}.$$

The behavior of $\bar{\rho}$ outside these regions is arbitrary.

Proof. Let

$$s := \min_{i \neq j} |\boldsymbol{p}_i - \boldsymbol{p}_j| > 0. \tag{123}$$

Choose

$$\epsilon_0 < \frac{s}{4}. \tag{124}$$

Fix $0 < \epsilon' < \epsilon_0$.

For each $i \in [N]$, define

$$K_i := \left\{(\boldsymbol{x}, \boldsymbol{y}) : |\boldsymbol{x} - \boldsymbol{p}_i| \leq \epsilon', \quad i \cdot \boldsymbol{y} \in \overline{H}_{\delta/2}\right\}, \tag{125}$$

where

$$\overline{H}_{\delta/2} := \left\{\boldsymbol{y} \in \mathbb{R}^d : \operatorname{dist}(\boldsymbol{y}, H) \leq \frac{\delta}{2}\right\}.$$

Since H is compact, each K_i is compact. Moreover, by Equation 124, the sets K_i are pairwise disjoint. Hence

$$K := \bigcup_{i=1}^{N} K_i \tag{126}$$

is compact.

Define $g : K \to \mathbb{R}^{d_{\text{out}}}$ by

$$g(\boldsymbol{x}, \boldsymbol{y}) := \rho(i \cdot \boldsymbol{y}), \qquad (\boldsymbol{x}, \boldsymbol{y}) \in K_i. \tag{127}$$

Because the sets K_i are pairwise disjoint compact sets and ρ is continuous, g is continuous on K.

We next show that the regions on which the network is required to output $\boldsymbol{u}_0$ are separated from K by a positive distance. For each $i \in [N]$, let

$$B_i := \{(\boldsymbol{x}, \boldsymbol{y}) : |\boldsymbol{x} - \boldsymbol{p}_i| \leq \epsilon', \quad i \cdot \boldsymbol{y} \notin H_{\delta}\}. \tag{128}$$

Consider $(\boldsymbol{x}, \boldsymbol{y}) \in B_i$ and $(\boldsymbol{x}', \boldsymbol{y}') \in K$.

If $(\boldsymbol{x}', \boldsymbol{y}') \in K_i$, then

$$\operatorname{dist}(i \cdot \boldsymbol{y}, H) \geq \delta, \qquad \operatorname{dist}(i \cdot \boldsymbol{y}', H) \leq \frac{\delta}{2}.$$

Since the distance-to-H function is 1-Lipschitz,

$$|\boldsymbol{y} - \boldsymbol{y}'| \geq \frac{\delta}{2i} \geq \frac{\delta}{2N}. \tag{129}$$

If instead $(\boldsymbol{x}', \boldsymbol{y}') \in K_j$ for some $j \neq i$, then

$$|\boldsymbol{x} - \boldsymbol{x}'| \geq |\boldsymbol{p}_i - \boldsymbol{p}_j| - |\boldsymbol{x} - \boldsymbol{p}_i| - |\boldsymbol{x}' - \boldsymbol{p}_j| \qquad \geq s - 2\epsilon'. \tag{130}$$

Consequently, if

$$\gamma := \min\left\{\frac{\delta}{2N}, s - 2\epsilon'\right\}, \tag{131}$$

then $\gamma > 0$ and

$$\operatorname{dist}(B_i, K) \geq \gamma \qquad \text{for every } i \in [N]. \tag{132}$$

We now construct a continuous function that agrees with g on K and is identically $\boldsymbol{u}_0$ sufficiently far from K. Let

$$d_K(\boldsymbol{x}, \boldsymbol{y}) := \operatorname{dist}((\boldsymbol{x}, \boldsymbol{y}), K).$$

Since K is compact, g admits a continuous extension $\widetilde{g} : \mathbb{R}^{2d} \to \mathbb{R}^{d_{\text{out}}}$. Define the continuous cutoff

$$\chi(\boldsymbol{x}, \boldsymbol{y}) := \begin{cases} 1, & d_K(\boldsymbol{x}, \boldsymbol{y}) \leq \gamma/3, \\ 2 - \dfrac{3d_K(\boldsymbol{x}, \boldsymbol{y})}{\gamma}, & \gamma/3 < d_K(\boldsymbol{x}, \boldsymbol{y}) < 2\gamma/3, \\ 0, & d_K(\boldsymbol{x}, \boldsymbol{y}) \geq 2\gamma/3, \end{cases} \tag{133}$$

and set

$$F(\boldsymbol{x}, \boldsymbol{y}) := \boldsymbol{u}_0 + \chi(\boldsymbol{x}, \boldsymbol{y})\big(\widetilde{g}(\boldsymbol{x}, \boldsymbol{y}) - \boldsymbol{u}_0\big). \tag{134}$$

Then F is continuous,

$$F(\boldsymbol{x}, \boldsymbol{y}) = g(\boldsymbol{x}, \boldsymbol{y}) \qquad \text{for } (\boldsymbol{x}, \boldsymbol{y}) \in K, \tag{135}$$

and

$$F(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{u}_0 \qquad \text{whenever } d_K(\boldsymbol{x}, \boldsymbol{y}) \geq \frac{2\gamma}{3}. \tag{136}$$

Choose a compact cube $Q \subset \mathbb{R}^{2d}$ whose interior contains the closed γ-neighborhood of K. Since F is continuous on Q, it is uniformly continuous. Thus, for any $\epsilon'' > 0$, there exists $h > 0$, which we choose additionally so that

$$h < \frac{\gamma}{3}, \tag{137}$$

such that

$$|F(\boldsymbol{x}, \boldsymbol{y}) - F(\boldsymbol{x}', \boldsymbol{y}')| < \epsilon'' \tag{138}$$

whenever both points lie in Q and their distance is smaller than h.

Triangulate Q into finitely many simplices of diameter smaller than h, and let P be the continuous piecewise-affine nodal interpolant of F on this triangulation. If $(\boldsymbol{x}, \boldsymbol{y})$ belongs to a simplex with vertices $\boldsymbol{v}_0, \ldots, \boldsymbol{v}_r$ and barycentric coordinates $\lambda_0, \ldots, \lambda_r$, then

$$P(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j=0}^{r} \lambda_j F(\boldsymbol{v}_j). \tag{139}$$

Using Equation 138,

$$|P(\boldsymbol{x}, \boldsymbol{y}) - F(\boldsymbol{x}, \boldsymbol{y})| \leq \sum_{j=0}^{r} \lambda_j |F(\boldsymbol{v}_j) - F(\boldsymbol{x}, \boldsymbol{y})| \qquad < \epsilon''. \tag{140}$$

In particular, Equations 135 and 140 imply

$$\sup_{(\boldsymbol{x}, \boldsymbol{y}) \in K} |P(\boldsymbol{x}, \boldsymbol{y}) - g(\boldsymbol{x}, \boldsymbol{y})| < \epsilon''. \tag{141}$$

We next verify that P is exactly equal to $\boldsymbol{u}_0$ sufficiently far from K. Suppose

$$d_K(\boldsymbol{x}, \boldsymbol{y}) \geq \gamma,$$

and let S be a simplex containing $(\boldsymbol{x}, \boldsymbol{y})$. For every vertex $\boldsymbol{v}$ of S, Equation 137 gives

$$\operatorname{dist}(\boldsymbol{v}, K) \geq d_K(\boldsymbol{x}, \boldsymbol{y}) - |\boldsymbol{v} - (\boldsymbol{x}, \boldsymbol{y})| \; > \gamma - h > \frac{2\gamma}{3}. \tag{142}$$

Thus, by Equation 136, $F(\boldsymbol{v}) = \boldsymbol{u}_0$ at every vertex of S, and therefore

$$P(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{u}_0. \tag{143}$$

The cube Q may be chosen so that the same argument applies to its boundary. We therefore extend P to all of $\mathbb{R}^{2d}$ by setting $P(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{u}_0$ outside Q. The resulting function remains continuous and piecewise affine with finitely many affine pieces.

Every continuous piecewise-affine function with finitely many pieces can be represented exactly by a finite ReLU network [1; 22]. Applying this representation coordinate-wise gives a finite ReLU MLP $G : \mathbb{R}^{2d} \to \mathbb{R}^{d_{\text{out}}}$ satisfying

$$G(\boldsymbol{x}, \boldsymbol{y}) = P(\boldsymbol{x}, \boldsymbol{y}). \tag{144}$$

Finally, define

$$\bar{\rho}(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) := G(\boldsymbol{x}, \boldsymbol{y}), \tag{145}$$

so that the network simply ignores $\boldsymbol{w}$ and $\boldsymbol{z}$.

Equation 141, together with Equations 127 and 125, gives Equation 121. Moreover, Equation 132 implies that every point in B_i has distance at least γ from K, so Equation 143 gives Equation 122. This completes the proof. □

B KV-CAT implementation details

Masked attention forward pass. Instead, at each layer group, 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

$$\mathcal{A}_t^\ell = \{j \leq t : m_j^{(\ell)} = 1\} \tag{146}$$

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.

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 $H^\ell = (h_1^\ell, \ldots, h_T^\ell)$, the router first computes

$$\tilde{h}_t = \mathrm{LN}(h_t^\ell), \qquad q_t = \phi(W_Q \tilde{h}_t), \quad k_t = \phi(W_K \tilde{h}_t), \quad r_t = W_V \tilde{h}_t, \qquad \phi(z) = \mathrm{ELU}(z) + 1. \tag{147}$$

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

$$S_t = \sum_{j \leq t} k_j r_j^\top, \qquad z_t = \sum_{j \leq t} k_j, \qquad a_t = W_O \frac{S_t^\top q_t}{q_t^\top z_t + \epsilon}. \tag{148}$$

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

$$u_t = \frac{W_P h_t^\ell}{\|W_P h_t^\ell\|_2}, \qquad w_t = \frac{h_t^\ell + \alpha a_t}{\|h_t^\ell + \alpha a_t\|_2}, \qquad p_t = \frac{1 - \langle u_t, w_t \rangle}{2}. \tag{149}$$

The binary routing decision is

$$m_t^\ell = \mathbf{1}\{p_t > \tau\}, \tag{150}$$

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 $\alpha = 0.001$, which gives $p_t = 1$ for almost all valid tokens, so training starts approximately from the dense model.

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

$$\begin{aligned}\mathcal{L} = \lambda_{\text{mask}} \frac{1}{T-1} \sum_{t=2}^{T} D_{\text{KL}}\big(\text{sg}\big[p_\theta^{\text{dense}}(\cdot \mid x_{<t})\big] \,\big\|\, p_\theta^{\text{mask}}(\cdot \mid x_{<t})\big) \\ + \lambda_{\text{anchor}} \frac{1}{T-1} \sum_{t=2}^{T} -\log p_\theta^{\text{dense}}(x_t \mid x_{<t}) + \lambda_{\text{budget}} \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \mathcal{B}_c.\end{aligned} \tag{151}$$

where sg stops gradients through the dense teacher. When using the router based KV sparsification policy, we use $\lambda_{\text{budget}} \neq 0$ and the budget term follows the load-balancing objective from Hwang et al. [24]. For target keep rate ρ, define

$$F_c = \frac{1}{T} \sum_{t=1}^{T} m_t^c, \qquad G_c = \frac{1}{T} \sum_{t=1}^{T} p_t^c, \qquad \mathcal{B}_c = \frac{F_c G_c}{\rho} + \frac{(1-F_c)(1-G_c)}{1-\rho}. \tag{152}$$

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

C Extended Experimental Details

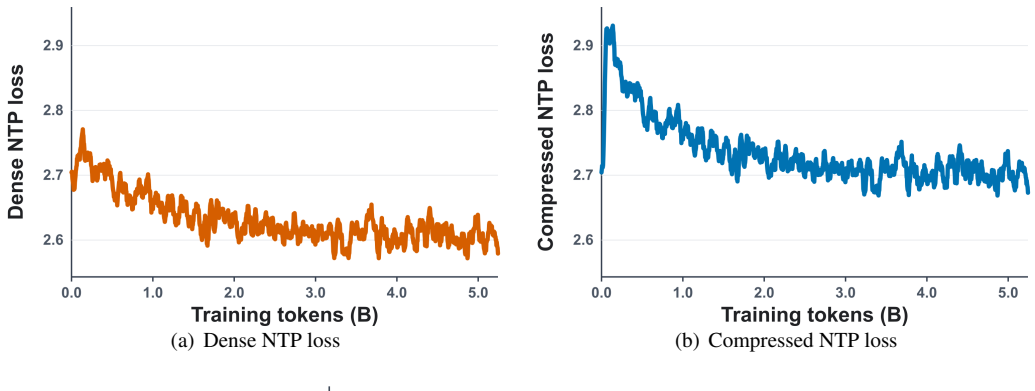
This section provides additional details for the empirical evaluation described in Section 5. We describe the continued-pretraining setup, including the data, architecture hyperparameters, and compression objective, and then give additional details on the post-hoc compression evaluations and experimental 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	1, 7, 13, 19	1, 8, 15, 22
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

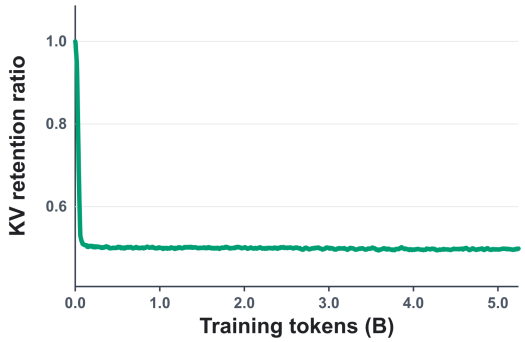
C.1 Continued pretraining runs

We train two model sizes, initialized from the public QWEN2.5-0.5B and QWEN2.5-1.5B [54] checkpoints, using continued pretraining on FineWeb-Edu [47] at a context length of 1024. The compression-aware objective is the one described in the method section: a compressed self-distillation 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.



(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%.

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 B) 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.

Continued-pretraining cost. KV-CAT is applied as a short continued-pretraining (CPT) stage on top of an already pretrained model. In our experiments, we continue pretraining for 5.24B tokens. For comparison, Qwen2.5 was originally pretrained on up to 18T tokens, making our CPT stage approximately 0.029% of the original pretraining token count. To quantify the computational overhead of KV-CAT during this stage, we benchmark standard CPT, KV-CAT with a random sparsification policy, and KV-CAT with our learned routing policy under matched settings using 8 H100 GPUs, an effective batch size of 131,072 tokens per update, and no gradient checkpointing. As shown in Table 6, the learned-router variant is $1.75\times$ slower per training step and uses $2.01\times$ the peak memory of standard CPT, while the random-policy variant incurs a substantially smaller 31.5% time overhead. Importantly, this additional computational cost is incurred only during the short CPT stage. At inference time, the routers and training-time masking are disabled, and the resulting model

Table 6: Computational cost of continued pretraining. We benchmark standard CPT and two variants of KV-CAT on Qwen2.5-0.5B using 8 H100 GPUs, with an effective batch size of 131,072 tokens per update and no gradient checkpointing.

Training	Time / step	Time overhead	Peak memory
Standard CPT	1.6925 s	–	32.86 GiB
Random-policy KV-CAT	2.2256 s	+31.5%	61.82 GiB
Learned-router KV-CAT	2.9647 s	+75.2%	66.15 GiB

is a standard transformer with no additional architectural or memory overhead. Thus, the one-time training cost is amortized across subsequent inputs, KV-cache budgets, and downstream compression methods.

C.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 [20]. The evaluation suite contains HellaSwag [64], WinoGrande [51], PIQA [5], OpenBookQA [41], ARC-Easy and ARC-Challenge [12]. The reported average is the unweighted mean over these six 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 post-hoc KV cache compression.

C.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-Edu. 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 evaluation protocol is shared across all compression methods; the methods differ only in how they construct the compact prefix cache.

Lightweight KV eviction. We evaluate five lightweight KV eviction methods at keep ratios $\rho \in \{0.1, 0.2, 0.4\}$. Unlike the optimization-based methods considered below, these approaches apply a simple, suffix-independent selection rule to the prefix cache, retaining a subset of the original KV states without optimizing new continuous representations. H_2O [65] retains high-importance heavy-hitter tokens, identified by accumulated attention mass, together with recent tokens. SnapKV [37] uses queries from an observation window at the end of the prefix to estimate per-head KV importance, pools neighboring attention scores to capture local clusters, and retains the highest-scoring positions. StreamingLLM [57] preserves the initial attention-sink tokens together with the most recent prefix tokens. KVzip [32] estimates KV importance through teacher-forced reconstruction of the prefix, assigning each KV pair the maximum attention it receives during reconstruction and evicting the lowest-scoring pairs. Finally, KeyDiff [46] scores keys according to their cosine similarity with a mean-key anchor and preferentially retains geometrically distinctive keys. For each method, we apply the resulting compressed prefix cache to the same 256-token suffix used by the dense model.

Attention-Matching. We use the AM-HighestAttnKeys Attention matching variant from Zweiger et al. [67]. For a keep ratio $\rho \in \{0.1, 0.2, 0.4\}$, 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,

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

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

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

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 probes complementary aspects of post-hoc KV cache compressibility. For the lightweight methods considered above, the compressed cache is constructed from the prefix alone, without access to the evaluation suffix; performance on the suffix therefore directly measures how well each method preserves the information in the prefix under a fixed cache budget. For the optimization-based methods, the compressor is additionally allowed to use supervision derived from the same prefix–suffix sequence. In this setting, we ask how effectively each checkpoint can be compressed under a matched optimization budget: for Attention Matching, we compare the quality of the resulting compact cache under the same fitting procedure, while for gradient-based optimization we additionally measure how many optimization steps are required to reach a given compression quality. Thus, the experiment evaluates both the quality of suffix-independent prefix compression and the ease with which more expressive compressors can optimize a compact representation of the cache.

C.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 sentence is

```
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 1,000 runs across these five depths.

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 a 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 C.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 exact-match retrieval accuracy: for each model, keep ratio, and compact-cache optimization step, we report the percentage of the 1,000 runs in which the compressed-prefix model generates exactly the passkey.

C.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.

We restrict this evaluation to Qwen2.5-1.5B with the gradient-based compressor, as the other model-size and compression-method combinations do not achieve sufficiently strong performance for a meaningful long-context evaluation. For each example, model, and keep ratio $\in \{0.1, 0.2, 0.5\}$, we optimize a compact KV cache for the example's context while keeping all model weights frozen. The optimization uses a reconstruction sequence as in Appendix C.4. As is 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) [3]. The KV 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.

D Additional Experimental Results

This section collects supplementary results that are complementary to Section 5. In Section D.1, we ablate the choice of the train-time sparsification policy used in KV-CAT. In Section D.4, we test whether the Attention Matching performance gains we observe in Section 5 generalize to prefix–suffix pairs from additional text corpora.

D.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 slots 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 [65] attention-mass baseline: it computes

Table 7: 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	52.2	**54.2**	69.8	**38.2**	58.5	**32.3**	**50.9**
	ROUTER	51.1	52.9	**70.8**	37.4	60.7	32.1	50.8
	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

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 B. 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 B. 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 sparsification policy in this ablation whose dense downstream accuracy is closest to the base model average. Its average score is 50.8, compared to 50.9 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 C.3. Table 8 shows that all three train-time sparsification policies improve Attention Matching performance compared to the matched continued-pretraining baseline for most reported metrics and budgets. 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 8: 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
10%	0.479	**0.422**	0.427	0.495	0.0367	**0.0321**	0.0356	0.0363	91.1%	**91.4%**	90.9%	90.8%
20%	0.566	**0.438**	0.547	0.594	0.0408	**0.0374**	0.0409	0.0396	90.6%	**90.8%**	90.4%	90.5%
40%	0.421	**0.360**	0.361	0.386	0.0311	**0.0272**	0.0283	0.0288	92.1%	**92.5%**	92.3%	92.4%

D.2 Uncompressed language modeling performance

As an additional check that KV-CAT does not degrade the model's native language-modeling capabilities, we evaluate both the base and KV-CAT checkpoints with no KV cache compression applied. We measure log-perplexity on a held-out 115K-token validation set, using the standard dense forward pass with 100% KV retention. As shown in Table 9, KV-CAT achieves slightly lower log-perplexity than the base model for both model sizes. These results provide additional evidence that the improvements in KV compressibility do not come at the expense of the model's uncompressed next-token prediction performance.

Table 9: Uncompressed language-modeling performance. We report log-perplexity on a 115K-token validation set with 100% KV retention. Lower is better.

Model	Base	KV-CAT
QWEN2.5-0.5B	2.63	**2.61**
QWEN2.5-1.5B	2.37	**2.36**

D.3 Additional metrics for compressed prefix NTP

We extend table 2 and report 2 additional similarity metrics between the suffix next token distribution under the full/compressed prefix: the KL divergence and Top-1 agreement. The extended results, including all three metrics across compression methods, keep ratios, and model sizes, are provided in Table 10.

Table 10: **Suffix completion under prefix KV compression.** Each example consists of a 768-token prefix and a 256-token suffix. We report ΔPPL, KL divergence, and Top-1 agreement between suffix predictions under the full and compressed prefix. Lower ΔPPL and KL are better; higher Top-1 is better. Each entry reports *Base / KV-CAT*.

		QWEN2.5-0.5B			QWEN2.5-1.5B		
Method	Keep	ΔPPL ↓	KL ↓	Top-1 ↑	ΔPPL ↓	KL ↓	Top-1 ↑
Attention Matching	10%	0.479 / **0.422**	0.0367 / **0.0321**	91.1 / **91.4**	0.664 / **0.320**	0.0601 / **0.0366**	89.7 / **90.7**
	20%	0.566 / **0.438**	0.0408 / **0.0374**	90.6 / **90.8**	1.385 / **0.536**	0.0915 / **0.0491**	88.5 / **90.2**
	40%	0.421 / **0.360**	0.0311 / **0.0272**	92.1 / **92.5**	0.659 / **0.259**	0.0581 / **0.0326**	90.9 / **92.4**
H_2O	10%	1.790 / **1.698**	0.1062 / **0.1000**	86.6 / **87.3**	1.319 / **1.142**	0.1043 / **0.0928**	86.6 / **87.6**
	20%	0.825 / **0.775**	0.0529 / **0.0490**	90.4 / **90.8**	0.618 / **0.495**	0.0575 / **0.0475**	89.9 / **91.2**
	40%	0.328 / **0.316**	0.0219 / **0.0204**	94.4 / **94.8**	0.332 / **0.184**	0.0271 / **0.0192**	93.1 / **94.3**
SnapKV	10%	1.906 / **1.799**	0.1148 / **0.1103**	86.6 / **86.9**	1.470 / **1.370**	0.1203 / **0.1092**	85.8 / **86.9**
	20%	0.919 / **0.833**	0.0585 / **0.0554**	90.7 / **91.4**	0.705 / **0.562**	0.0612 / **0.0510**	89.9 / **91.1**
	40%	0.330 / **0.297**	0.0262 / **0.0246**	**94.5** / 94.4	0.373 / **0.224**	0.0314 / **0.0221**	93.3 / **94.3**
KVzip	10%	2.746 / **2.389**	0.1663 / **0.1559**	83.3 / **83.9**	1.976 / **1.654**	0.1625 / **0.1413**	83.2 / **84.7**
	20%	1.565 / **1.333**	0.0987 / **0.0907**	87.0 / **87.6**	1.108 / **0.821**	0.0979 / **0.0774**	86.7 / **88.4**
	40%	0.633 / **0.508**	0.0428 / **0.0367**	91.9 / **92.2**	0.469 / **0.227**	0.0435 / **0.0286**	90.9 / **92.7**
StreamingLLM	10%	2.025 / **1.941**	0.1231 / **0.1184**	86.3 / **86.4**	1.572 / **1.483**	0.1301 / **0.1189**	85.0 / **86.0**
	20%	1.290 / **1.192**	0.0771 / **0.0753**	89.5 / **89.6**	1.067 / **0.913**	0.0862 / **0.0768**	88.0 / **89.2**
	40%	0.502 / **0.471**	0.0343 / **0.0336**	**93.2** / **93.2**	0.512 / **0.393**	0.0426 / **0.0351**	91.7 / **93.0**
KeyDiff	10%	3.986 / **3.426**	0.2251 / **0.2009**	79.6 / **80.6**	2.846 / **2.543**	0.2153 / **0.1932**	80.0 / **81.2**
	20%	2.403 / **2.053**	0.1434 / **0.1268**	82.9 / **83.7**	1.651 / **1.399**	0.1362 / **0.1169**	83.6 / **84.8**
	40%	1.037 / **0.933**	0.0646 / **0.0568**	87.8 / **88.8**	0.697 / **0.510**	0.0596 / **0.0488**	88.4 / **89.5**

D.4 Cross-domain attention matching evaluation

We test whether the Attention Matching optimization gains observed for KV-CAT transfer to prefix–suffix pairs from additional textual domains. We evaluate Attention Matching KV cache compression on validation examples from three held-out corpora: WikiText-103 [39], PG-19 [49], 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 C.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 8 of 9 settings; the only exception is arXiv at 40% keep ratio, where ΔPPL increases even though KL and top-1 agreement still improve.

D.5 Additional KV keep-ratio curves

Figure 4 extends the main-body optimization curves in Figure 2 with three additional KV keep ratios: 15%, 30%, and 40%.

Table 11: **Attention Matching cross-domain completion under prefix-cache compression.** We compare the base QWEN2.5-1.5B model and the KV-CAT-trained QWEN2.5-1.5B checkpoint, on held-out prefix–suffix pairs from different domains. Each example contains a 768-token prefix and a 256-token suffix; Attention Matching compresses only the prefix KV cache. Metrics are computed relative to each model's native dense full-prefix suffix distribution, so lower ΔPPL and KL and higher top-1 agreement are better.

Corpus	Keep ratio	Variant	ΔPPL (↓)	KL (↓)	Top-1 (↑)
arXiv	10%	Base	1.414	0.1593	84.1%
		KV-CAT	**1.176**	**0.1340**	**85.6%**
	20%	Base	1.747	0.1801	83.4%
		KV-CAT	**1.243**	**0.1488**	**84.7%**
	40%	Base	**1.526**	0.1609	85.0%
		KV-CAT	2.842	**0.1599**	**86.1%**
PG-19	10%	Base	1.096	0.0702	87.7%
		KV-CAT	**0.736**	**0.0457**	**88.8%**
	20%	Base	1.580	0.0863	86.7%
		KV-CAT	**1.123**	**0.0604**	**87.9%**
	40%	Base	1.245	0.0681	88.7%
		KV-CAT	**0.587**	**0.0394**	**90.1%**
WikiText-103	10%	Base	1.905	0.0812	89.3%
		KV-CAT	**1.242**	**0.0498**	**91.5%**
	20%	Base	2.411	0.0945	88.6%
		KV-CAT	**1.381**	**0.0591**	**90.8%**
	40%	Base	1.985	0.0808	89.8%
		KV-CAT	**1.269**	**0.0481**	**91.7%**

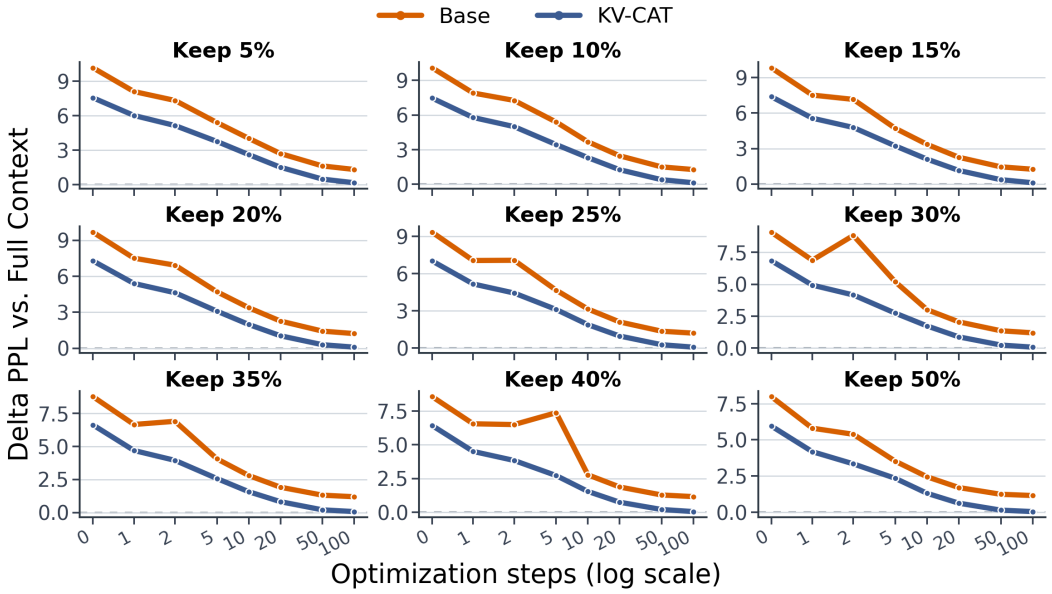


Figure 4: **KV-CAT speeds up gradient-based KV cache compression.** We plot the gap in suffix perplexity under full/compressed-prefix inference throughout gradient-based KV cache optimization. Each panel fixes a different KV keep ratio. Across ratios, the KV-CAT checkpoint achieves a comparable ΔPPL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

D.6 Needle-in-a-haystack retrieval

Table 12 supplements the NIAH results in Table 3 with variation across seeds, using the evaluation protocol in Appendix C.4.

Table 12: **Needle retrieval from compressed haystack.** Exact-match success rate (%), reported as mean ± standard deviation across five seeds with 200 prompts per seed.

	QWEN2.5-0.5B		QWEN2.5-1.5B	
Keep	Base	KV-CAT	Dense-CPT	KV-CAT
Full	**99.2** ± 0.8	96.4 ± 0.5	87.6 ± 1.5	**90.8** ± 3.2
10%	**15.7** ± 1.3	14.5 ± 1.4	19.7 ± 0.9	**20.4** ± 0.8
20%	**15.6** ± 1.8	14.8 ± 1.6	22.7 ± 1.8	**25.5** ± 1.8
30%	25.8 ± 1.6	**30.5** ± 4.2	42.5 ± 1.5	**46.5** ± 1.5
40%	27.6 ± 1.9	**38.3** ± 1.8	48.0 ± 2.3	**52.6** ± 1.9
50%	33.4 ± 2.3	**50.5** ± 1.8	59.4 ± 1.3	**65.9** ± 3.4

E Limitations

Our approach relies on continued pretraining with additional routing modules, introducing non-trivial training overhead and potentially limiting applicability in low-resource settings. The routing mechanism and training objective also add implementation complexity, which may hinder integration into existing production systems. Finally, our evaluation is limited to a small set of model sizes and tasks, and it remains to be seen how well the method generalizes to larger-scale models or different domains.

F Broader Impact

Broader Impact. This work is primarily methodological, aiming to improve the efficiency of transformer-based language models by enabling more effective KV cache compression. On the positive side, our approach can reduce memory and compute requirements for long-context inference, making large models more accessible and energy-efficient, and enabling their deployment in resource-constrained settings. Potential negative impacts are indirect: by improving the efficiency of existing LLMs, our method may lower the cost of deploying such models at scale, which could amplify both beneficial and harmful downstream applications. However, our method does not introduce new capabilities or alter model behavior beyond improving compatibility with compression, and therefore does not create new misuse vectors beyond those already present in the underlying models. Mitigation strategies largely align with existing practices for responsible LLM deployment, including monitoring, access control, and adherence to usage policies of the underlying models.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made match the paper's experimental and theoretical results.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: Limitations of our methods are mentioned throughout the paper, and discussed in depth in Appendix E.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All theoretical results are rigorously proved in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

Justification: All information needed to reproduce our experimental results is provided in the Appendix in Sections C and B.

Guidelines:

- The answer [N/A] means that zthe paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The codebase for the experiments and method is complex and currently not in a state that supports straightforward external use. We are committed to releasing a cleaned, documented, and fully reproducible version of the code upon publication.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.

- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the main experimental details in Section 5, while full details are provided in Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Our experiments involve large-scale training of LLMs with up to 1.5B parameters, making repeated runs computationally prohibitive given the compute resources available to us. As is common in the literature, we do not report error bars for such experiments. We note that the evaluated metrics are typically stable across runs for models of this scale.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Details on compute resources used in this paper are provided in Appendix C.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: This work is purely methodological and involves no human subjects or sensitive data, and therefore raises minimal ethical concerns.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The broader impact of the paper is discussed in Appendix F

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our work proposes a method to make existing open-source LLMs more compatible with KV cache compression methods. It improves efficiency but does not introduce new capabilities or increase the risk of misuse, and therefore does not require additional safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All external assets used in this work, including pretrained models, datasets, and prior methods, are properly credited in the text and references. We adhere to their respective licenses and terms of use, and only use publicly available resources in accordance with their stated conditions.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.

- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.