VFOLD: SYMMETRY-AWARE CROSS-LAYER VALUE CACHE COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

While caching key-value (KV) states accelerates Large Language Model (LLM) decoding, this cache can dominate memory usage at long context lengths. One solution is to compress this memory by exploiting inter-layer cache similarities. However, most existing techniques necessitate architectural changes to LLMs and incur substantial overhead. In this work, we propose a symmetry-aware value cache merging strategy that reduces cache memory while avoiding both harmful performance degradation and architectural overhead during decoding. Furthermore, we show that this approach can be exploited alongside existing cache compression techniques, composing with high-ratio quantization or key cache pruning to reach compression ratios that neither method reaches alone, with minimal additional cost. Ultimately, our findings reveal a major source of underutilized capacity in the value cache, offering a simple yet highly effective direction for scaling context windows under memory constraints.

1 INTRODUCTION

Expanding context windows has unlocked significant new capabilities in LLMs, allowing them to reason over multi-document (Bai et al., 2024; An et al., 2024) or repository-level contexts (Jimenez et al., 2024; Bairi et al., 2024). However, these advancements come at a cost; long contexts shift the bottleneck of autoregressive decoding from compute to memory. This shift is driven primarily by the presence of the key-value (KV) cache, which stores key and value vectors from attention for prior tokens, in order to prevent their expensive recomputation. At sufficiently long contexts, this cache can far surpass the memory requirement of the LLM parameters (Pope et al., 2023). For example, for 16GB Llama-3.1-8B (fp16), a batch of 8 inputs at 16K token length also incurs a 16 GB KV cache size, quickly meeting the memory demands of the weights themselves (Grattafiori et al., 2024).

Due to this cache memory cost, multiple techniques have been proposed to try to reduce its footprint. Approaches to KV cache compression can be grouped by the mechanism they employ. *Eviction* methods reduce the cache by selecting a subset of paired key/value items to retain while discarding the others (Zhang et al., 2023; Xiao et al., 2024; Devoto et al., 2024; Kim et al., 2025). *Quantization* methods retain caches but store keys and values at reduced bit width (Liu et al., 2024b; Zandieh et al., 2025; Shutova et al., 2025). Methods have also exploited low-rank representations of KV caches, and store only rank-reduced representations (Wang et al., 2025; Chang et al., 2026). Finally, some methods have looked at sharing caches across layers, whether at initialization or after training (Brandon et al., 2024; Wu & Tu, 2024; Liu et al., 2024a; Sun et al., 2024; Lin et al., 2026).

Cross-layer cache sharing, however, has largely been investigated in settings that only allow LLM training from scratch, leaving open the question of how best to apply these strategies to pre-existing models. The few methods that do apply to pre-existing models are typically motivated by a common empirical observation: while key and value vectors across layers have low raw pairwise similarity, they are more closely aligned under measures that account for a common subspace (Kornblith et al., 2019; Wang et al., 2025; Chang et al., 2026), so a single shared basis can represent several layers' caches with little reconstruction error. Existing cross-layer methods frequently realize this shared basis in ways that trade one cost for another. Low rank approaches (Wang et al., 2025; Chang et al., 2026) store caches in a compressed low-rank form, forcing "always-on" compression, which requires modifications to attention and the reconstruction of vectors during decoding. Pre-existing

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

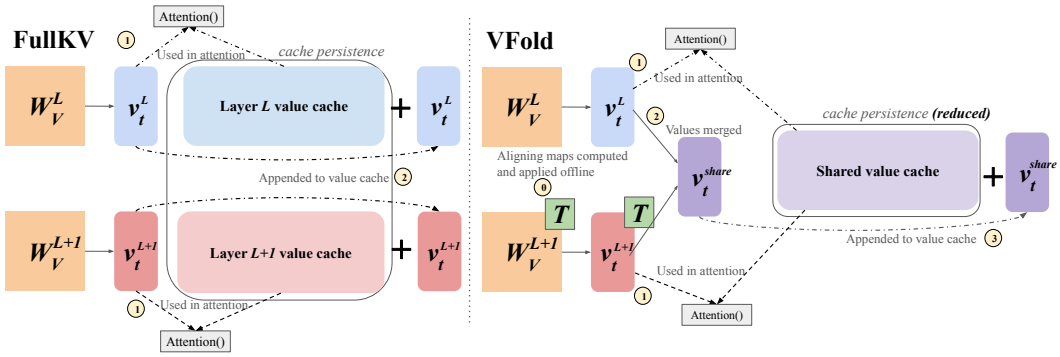


Figure 1: Overview of VFOLD. In the full cache setting (left), individual value caches are stored for layers L and $L+1$. Each layer's new value vector for the tth token is appended to its respective cache. In the VFOLD setting (right), we first offline fit and absorb a function-preserving linear map T and its inverse into W_V and W_O (not shown), respectively (step 0). The resulting value vectors are used in attention (step 1) unperturbed, and then merged into a single value vector (step 2), that is then appended to a shared inter-layer cache (step 3), resulting in memory savings.

cache merging-based approaches avoid this overhead, but pay in accuracy; MiniCache interpolates caches without re-projection, but degrades heavily on long-context benchmarks (Liu et al., 2024a).

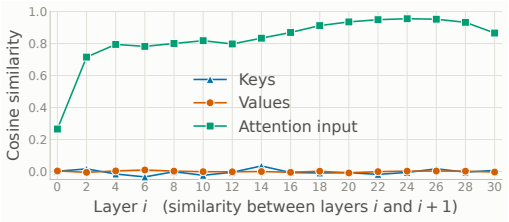
To avoid these costs, we propose VFOLD: a simple, query-agnostic method that compresses the KV cache of a pre-trained model by aligning and sharing value caches across adjacent layers (Figure 1). Our contributions are threefold. **(1) Analysis:** We demonstrate that although adjacent layers' keys and values show high similarity under Centered Kernel Alignment (CKA), only values are amenable to cross-layer averaging (Kornblith et al., 2019). **(2) Method:** We exploit a symmetry within multi-headed attention to fold learned per-head permutations and Canonical Correlation Analysis (CCA) alignment maps directly into attention weights to better align a layer's value cache in terms of a reference layer's value cache basis. This model, modified in-place, computes exactly the same function; our method applies without additional fine-tuning, parameters, or changes to attention until cache approximation occurs. **(3) Results:** Halving the value cache via this method (25% total reduction) retains >98% of full-cache performance on RULER (Hsieh et al., 2024) and LongBench (Bai et al., 2024) across three models, outperforming prior cross-layer methods with the lowest decoding overhead and minimal prefill overhead. We also demonstrate that this method composes with 4-bit KIVI and ThinK key pruning for additional compression (Liu et al., 2024b; Xu et al., 2025).[1]

2 Related Work

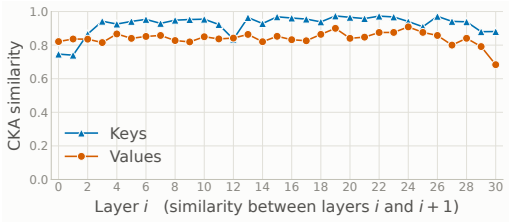
Cross-layer KV cache compression Prior work has explored methods that share KV caches across model layers in order to reduce total cache memory consumption. Many of these methods create cache sharing patterns at model initialization and pre-train with these patterns, including Cross-Layer Attention (Brandon et al., 2024), You Only Cache Once (Sun et al., 2024), Layer-Condensed KV Cache (Wu & Tu, 2024), and FusedKV (Lin et al., 2026). However, these methods, as proposed, do not apply to pre-existing models trained with standard per-layer caching.

Some work has indeed investigated the question of reducing cache memory via cross-layer sharing techniques on pre-existing models. For example, Liu et al. (2024a) propose MiniCache, a cross-layer merging technique utilizing spherical interpolation (SLERP) to combine adjacent caches while protecting high importance tokens, but not without a notable cost in accuracy. Yang et al. (2024) propose KVSharer, which directly shares dissimilar layers' caches for both prefill and generation. Other related techniques utilize cross-layer low-rank compression on pre-trained models: xKV (Chang et al., 2026) compresses after prefill by computing, per request, a single basis across KV caches for a group of layers. CommonKV instead operates on weights by applying a joint SVD across key and value projection matrices of a group, before merging the resulting caches (Wang et al., 2025). Finally,

[1]We will release full source code upon publication.



(a) Summary of cosine similarities for adjacent model layers' structures. While inputs to attention layers are quite similar, likely in large part due to residual connections, adjacent layers' key and value caches have nearly zero cosine similarity.



(b) Summary of CKA similarities between adjacent layers keys and value caches. As compared to cosine similarity, CKA similarity is much higher, indicating similarity within latent space rather than direct embedding similarity.

Figure 2: Adjacent KV layer similarities for Llama-3.1-8B-Instruct on GSM8K show near-zero cosine but high CKA similarity.

recent work has examined using small predictor modules trained on example data to predict key and value vectors items based on prior KV items across layers. AQUA-KV employs this for quantization (Shutova et al., 2025), and EchoKV employs this for direct compression (Ji et al., 2026). Our method departs from these predictor methods by avoiding architectural modifications with directly foldable alignment maps.

Symmetry-aware reparameterizations Separately, prior work has investigated exploiting weight-space symmetries in Transformers, by folding function-preserving maps into weights in order to achieve various downstream objectives. For example, SliceGPT folds truncated rotation matrices into LLMs to achieve structured pruning (Ashkboos et al., 2024a); similarly, QuaRot and SpinQuant fold full rotation matrices into LLMs in order to redistribute activation outliers before downstream activation quantization is applied (Ashkboos et al., 2024b; Liu et al., 2025). RotateKV applies a similar outlier-aware rotation framework specifically to the KV cache (Su et al., 2025). These methods generally focus on orthogonal maps to preserve vector norms. Extensive work in model merging has exploited various forms of symmetry-aware reparameterizations for aligning several models (Ainsworth et al., 2023; Horoi et al., 2024; Stoica et al., 2024; Verma & Elbayad, 2024). We adopt similar mapping principles to prepare models for downstream cache compression rather than quantization or model merging objectives.

3 Preliminary Analysis and Motivation

Inter-layer KV cache similarity We first corroborate prior work demonstrating that despite having low cosine similarity, adjacent layer caches have high CKA similarity, suggesting a similar underlying basis (Kornblith et al., 2019). CKA (Centered Kernel Alignment) is invariant to rotations and scaling, allowing for similarity measurements that do not depend on an arbitrary basis for comparison. For this analysis, we use Llama-3.1-8B-Instruct as an example model (Grattafiori et al., 2024), and GSM8K as an example dataset (Cobbe et al., 2021). For each layer, we collect layer inputs, keys, and values on GSM8K, and compute cosine similarities for each adjacent pair, found in Figure 2a. Despite having high input similarity, corresponding keys and values across layers do not have notable cosine similarity. For CKA similarities, we simply swap out the metric given the same setup for measuring cosine similarity, and report key and value results in Figure 2b.[2] As seen in the figure, both keys and values have high CKA similarity across layers, corroborating prior results, and motivating cross-layer cache compression. This finding is similar to those motivating CommonKV and xKV (Wang et al., 2025; Chang et al., 2026). This result suggests that similarity between keys or values between adjacent layers exists, but only up to a change of basis.

Value stability compared to keys When the KV cache is compressed in the token dimension, as in eviction, each key/value pair is removed as a unit, so key and value caches are reduced identically. Non-eviction methods are not bound by the same constraint; they can compress only one of the

[2]We use 10,000 tokens for computational feasibility with CKA.

caches as in ThinK (keys) (Xu et al., 2025), or treat keys and values asymmetrically, as in KIVI (Liu et al., 2024b), AsymKV (Tao et al., 2025), and EchoKV (Ji et al., 2026). Since we are interested in exploiting cross-layer cache redundancy, we investigate how amenable both keys and values are to simply being averaged across layers. We again use Llama-3.1-8B-Instruct and GSM8K, and ask how much performance is lost when averaging keys versus when averaging values. This means that for each non-overlapping layer pair in the model, two caches are averaged and then stored into memory once, but retrieved by two separate layers for future computations. For averaging, we do not merge the first 4 tokens and a buffer of the last 128 tokens, as standard in prior literature (Xiao et al., 2024; Liu et al., 2024b; Staniszewski & Lancucki, 2026), avoiding their inclusion in cache compression to preserve critical performance. We perform 8-shot Chain-of-Thought completion on GSM8K with greedy decoding (Wei et al., 2022), and report average accuracy across the averaging and full cache settings in Table 1.

From our results, we find values to be far more amenable to merging, despite both keys and values displaying high CKA similarity as seen in Figure 2b. This asymmetry can potentially be explained by the differing roles keys and values play in attention; while perturbing keys changes how tokens are attended to, which is then amplified by softmax, value perturbation only changes what is retrieved, which then is subject to a linear combination across all prior values.

Table 1: GSM8K accuracy when averaging keys vs. values across adjacent layers.

Setting	Acc.
Full cache	80.7%
Avg. keys	2.7%
Avg. values	79.5%

Attention symmetries and exact value recovery Given the observed high inter-layer value cache CKA similarity, we are interested in a low-overhead alignment map to further improve values' mergeability. To do this, we exploit a natural linear invariance in attention between the value projection (W_V) and output projection (W_O), also described in prior attention symmetry work (Zhang et al., 2025).

For a given attention head h with inputs $X \in \mathbb{R}^{n \times d_{\text{model}}}$, values $V^h = XW_V^h$, and attention matrix A^h, the head's output is

$$\text{Output of Head } h = (A^h V^h) W_O^h \in \mathbb{R}^{n \times d_{\text{model}}} \tag{1}$$

which demonstrates the aforementioned linear relationship. Given this relationship, for any head, we can insert map $T_h \in \text{GL}_{d_h}(\mathbb{R})$[3] between values and the output matrix:

$$[A^h (X W_V^h) T_h][T_h^{-1} W_O^h] \tag{2}$$

Due to matrix associativity, we can fold T_h into W_V^h and T_h^{-1} into W_O^h entirely offline. Once the transform is applied, the model can run decoding without additional inference or memory cost. In the case of grouped query attention, we apply T^{-1} to all W_O segments corresponding to the same KV head (Ainslie et al., 2023). Critically, this folding technique is in contrast with prior work on low-rank cache compression, where shared SVD low-rank projections drive the aligning of separate layers, at the cost of architectural modification. These modifications generally result in every new token being compressed. In our case, full value vectors are recovered after folding, whereas in these low-rank substitutions, exact value vectors are not recoverable due to the "always-on" nature of these methods (Saxena et al., 2024; Chang et al., 2025; Wang et al., 2025).

This same invariance does not apply to the key cache, due to the presence of RoPE in modern architectures (Su et al., 2024). Because the same linear structure does not apply to W_K with RoPE, any such map would have to commute with the RoPE rotation. This heavily restricts degrees of freedom of the map and conflicts with RoPE being commonly fused into attention kernels for speed (Ye et al., 2025).

4 VFOLD

Building on the empirical results and symmetry described in Section 3, we propose our value cache compression method VFOLD. Our approach consists of offline weight reparameterization followed by runtime cache sharing that enables the cross-layer compression.

[3]This is the general linear group of degree d_h.

Computing alignment maps To maximize the similarity of adjacent value caches to prepare them for downstream merging, which Table 1 shows is already promising without alignment, we construct alignment maps using Canonical Correlation Analysis (CCA) on input tokens (Horoi et al., 2024). For a base layer ℓ and subsequent layer $\ell+1$, CCA finds projections U_ℓ and $U_{\ell+1}$ that project each cache, respectively, into a space with maximum correlation. Rearranging, we define map $T = U_{\ell+1}U_\ell^{-1}$, which is applied to $\ell+1$ alone.

While the above map only applies to a single KV head in multi-head or grouped-query attention, we can extend a map to full value vectors. In order to maintain attention invariance, dimensions from different KV heads cannot be mixed; therefore, we optimally pair heads between layers ℓ and $\ell+1$ based on their value states. Then, for every potential head pairing (i, j) between layer ℓ and $\ell+1$, we compute CCA map T_{ij}, and define the alignment cost via the Frobenius error:

$$C_{ij} = \text{cost}(i,j) = ||V_\ell^{h_i} T_{ij}^{-1} - V_{\ell+1}^{h_j}||_F \quad (3)$$

We then use the Hungarian algorithm to find the optimal head pairing, or permutation π, that minimizes the total alignment cost across all n_{kv} KV heads:

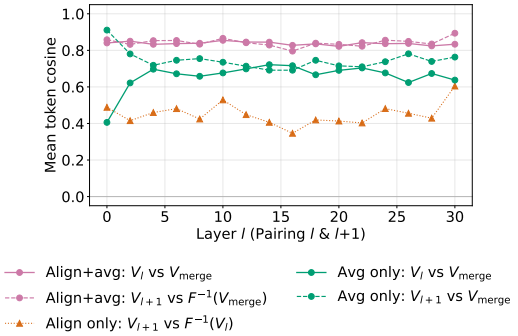


Figure 3: Similarity between combined caches and original value caches. We construct the aligned average as $V_{\text{merge}} = \frac{1}{2}(V_\ell + F(V_{\ell+1}))$ and compare it with V_ℓ, for both $F = I$ (unaligned) and CCA. We also map the merged cache back to the coordinate system of layer $\ell+1$ and compare $F^{-1}(V_{\text{merge}})$ with $V_{\ell+1}$. The aligned average achieves cosine similarities close to 0.85 on held-out data.

$$\pi = \arg\min_{\Pi_{n_{\text{kv}}}} \sum_{i=1}^{n_{\text{kv}}} C_{i,\pi(i)} \quad (4)$$

Then, defining $T = \text{blockdiag}(T_1, \ldots, T_{n_{\text{kv}}})$, we convert π to its matrix form P. We apply the maps P to heads of attention weights W_Q, W_K and W_V to reorder them, and then apply T specifically to W_V after heads are re-ordered. More details on these mappings are in Appendix A. Then, to undo this operation, we apply T^{-1} and P^T (applied to heads) to W_O. A summary of this map application appears in Figure 4.[4] More details on CCA and its invertibility are in Appendix B.

Aligned cache cosine similarity To demonstrate the effectiveness of our mapping algorithm, we compute the cosine similarities of aligned and averaged caches with their addends before and after alignment. We compute example caches on Wikitext-2 (Merity et al., 2017),[5] and test our maps on 50,000 tokens of Wikitext-2 and held-out GSM8K data. The aligned and averaged caches are computed as the following for layers ℓ and $\ell+1$, where $F(V) = VPT$

$$V_{\text{merge}} = \frac{1}{2}(V_\ell + F(V_{\ell+1})).$$

We then measure its similarity to both original caches. For layer ℓ, we directly compare V_{merge} with V_ℓ. For layer $\ell+1$, we map the merged cache back to the original coordinate system and compare $F^{-1}(V_{\text{merge}})$ with $V_{\ell+1}$. As seen in Figure 3, the merged cache remains highly similar to both source caches, reaching cosine similarities of approximately 0.85 on the held-out data. Similar results for in-distribution data are in Appendix C. Note that before, from Figure 2a, adjacent value caches have near-zero cosine similarity without alignment.

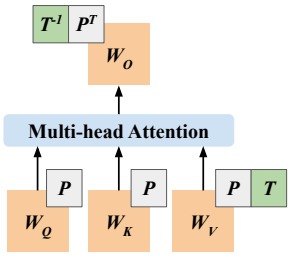


Figure 4: In VFOLD, we apply two main invertible maps to prepare for cache merging. P permutes the ordering of the KV heads, and propagates to all relevant attention weight matrices. T, block-diagonally shaped, transforms each head to be more similar to its corresponding head in a reference layer.

[4]For QKNorm, each head uses the same RMSNorm weights, which does not interact with this permutation. RoPE is also applied identically across heads.

[5]We use 256 batches of length 2048.

Table 2: RULER (16K) results for KV cache compression methods at 25% total reduction, matching VFOLD's 50% value cache reduction. **Best** average results are bolded, second-best are underlined.

	Needle In A Haystack (NIAH)								Synthetic			QA		
Method	**S1**	**S2**	**S3**	**MK1**	**MK2**	**MK3**	**MQ**	**MV**	**CWE**	**FWE**	**VT**	**QA1**	**QA2**	**Avg**
Llama-3.1-8B-Instruct														
Full KV	*100.0*	*100.0*	*100.0*	*99.80*	*100.0*	*99.60*	*99.25*	*98.75*	*81.76*	*91.13*	*98.96*	*79.40*	*53.60*	*92.48*
MiniCache	58.20	74.40	14.40	65.20	56.20	8.80	32.70	28.35	1.28	44.47	47.04	66.40	46.00	41.80
MiniCache-V	96.60	92.80	84.40	84.80	63.40	9.20	75.35	80.45	17.22	63.00	49.92	66.00	47.80	63.92
CommonKV	95.80	96.80	89.40	97.40	84.80	57.00	94.45	85.00	5.78	89.87	81.32	73.20	49.20	76.92
VFOLD-NAIVE	97.60	94.60	96.80	96.60	99.20	96.60	89.40	79.60	2.20	80.27	73.96	76.40	51.00	79.56
VFOLD	100.0	100.0	100.0	99.60	100.0	98.20	97.50	96.20	75.26	88.60	98.68	79.00	51.80	**91.14**
Mistral-Small-3.1-24B-Instruct														
Full KV	*100.0*	*99.80*	*100.0*	*99.80*	*100.0*	*99.40*	*100.0*	*99.95*	*96.48*	*98.73*	*100.0*	*85.00*	*68.00*	*95.94*
MiniCache	46.40	8.60	0.00	5.60	0.80	0.00	2.95	4.45	22.86	84.80	84.96	60.00	47.00	28.34
MiniCache-V	99.80	100.0	98.80	97.60	97.20	60.60	93.85	96.00	24.42	86.73	97.64	73.40	58.80	83.45
CommonKV	100.0	99.80	100.0	99.80	99.80	99.20	99.90	99.95	94.02	93.87	100.0	83.00	65.00	94.95
VFOLD-NAIVE	100.0	99.80	100.0	99.80	100.0	99.40	99.75	97.90	51.32	92.53	99.24	80.80	64.20	91.13
VFOLD	100.0	100.0	100.0	99.80	100.0	99.40	99.85	99.75	91.46	96.93	100.0	83.80	65.60	**95.12**
Qwen3-8B														
Full KV	*100.0*	*100.0*	*99.80*	*99.80*	*100.0*	*99.80*	*99.85*	*99.55*	*83.76*	*93.80*	*100.0*	*75.40*	*59.00*	*93.14*
MiniCache	4.20	0.20	0.00	0.80	0.00	0.00	0.15	0.30	1.82	34.80	2.16	20.20	15.80	6.19
MiniCache-V	97.20	95.80	94.60	95.60	67.00	19.00	73.85	79.20	29.88	83.27	91.32	61.00	49.60	72.10
CommonKV	100.0	100.0	97.60	98.60	98.40	91.40	99.45	96.30	59.98	93.73	99.24	66.20	54.00	88.84
VFOLD-NAIVE	99.80	100.0	98.00	99.80	93.20	16.20	99.30	95.65	32.42	71.47	95.68	70.00	53.40	78.84
VFOLD	100.0	100.0	99.40	99.60	99.60	98.00	99.75	99.10	73.46	92.00	99.88	74.00	56.40	**91.63**

Direct averaging without alignment achieves a cosine similarity of approximately 0.7, which is higher than the similarity obtained by reconstructing one cache from the other using alignment alone, and close to the theoretical similarity for uncorrelated random vectors. Nevertheless, aligned merging provides substantially higher similarity to both source caches, compared to either averaging or alignment alone, motivating the use of both in our method.

While we describe VFOLD for groups of two layers, we extend this method to larger groups of layers in Appendix D. As caches are materialized, layer-by-layer, we combine them, which reduces the realized peak memory during the forward pass. We temporarily store V_ℓ as the merged cache, and then overwrite it with V_{merge} after $V_{\ell+1}$ is materialized. We employ a similar running mean for larger group sizes as well. Because the folded model is function-preserving, VFOLD only introduces error when a value, after being used in attention, is merged into the shared cache. The current step is always computed with full values, which is not possible in many architecture-modifying methods, like low-rank methods.

5 EXPERIMENTAL SETTINGS

Models We test our method using 3 models: Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Mistral-Small-3.1-24B-Instruct (Liu et al., 2026), and Qwen3-8B (Yang et al., 2025), containing 32, 40, and 36 layers, respectively. All models use Grouped-Query Attention; Qwen3-8B additionally includes Query-Key Normalization (Henry et al., 2020).

Evaluation To evaluate capability retention under compression, we use both the LongBench and RULER evaluation benchmarks (Bai et al., 2024; Hsieh et al., 2024). LongBench contains real-world long-context tasks covering six task areas: single- and multi-document QA, summarization, few-shot learning, synthetic tasks, and code. RULER is a suite of synthetic long context tasks including Needle-in-a-Haystack tasks and aggregative tasks; we use its 16K sequence length version.

Baselines We compare our method to prior work that also employs cross-layer cache sharing techniques. MiniCache is one such method, where adjacent layer key and value caches are combined using spherical interpolation, while keeping important tokens unmerged, and storing norm informa-

Table 3: LongBench results for KV cache compression methods at 25% KV reduction, matching VFOLD's 50% value cache reduction. **Best** average results are bolded, second-best are underlined.

Methods	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot			Synth.		Code		
	Nrtv	Qasp	MF-en	Hotpot	2Wiki	Musiq	Gov	QMS	MN	TREC	TQA	SAM	PC	PRe	LCC	RB-P	Avg
Llama-3.1-8B-Instruct																	
Full KV	*31.3*	*46.8*	*56.6*	*58.1*	*48.9*	*32.5*	*34.5*	*25.3*	*27.0*	*72.5*	*91.7*	*43.7*	*8.6*	*100.0*	*65.1*	*58.3*	*50.0*
MiniCache	20.9	31.7	40.4	40.9	24.5	20.4	23.0	21.9	22.9	66.0	79.3	36.1	11.3	86.9	47.5	46.6	38.8
MiniCache-V	26.2	40.4	48.8	49.2	36.1	27.0	21.0	22.5	24.0	61.5	91.0	41.4	3.7	75.9	44.3	42.4	41.0
CommonKV	30.3	40.9	54.5	55.4	47.8	30.2	26.0	24.1	25.0	62.0	90.0	42.9	5.1	99.5	63.7	58.3	47.3
VFOLD-NAIVE	32.5	43.1	55.6	58.3	47.0	31.8	25.1	24.5	23.3	69.5	91.3	40.4	8.6	100.0	62.7	54.6	48.0
VFOLD	32.0	45.1	57.4	57.9	48.0	31.9	31.5	25.1	25.2	72.5	91.5	43.0	7.1	99.5	65.2	57.4	**49.4**
Mistral-Small-3.1-24B-Instruct																	
Full KV	*37.5*	*51.5*	*54.8*	*68.9*	*65.7*	*50.2*	*33.1*	*25.0*	*24.7*	*77.0*	*93.2*	*50.3*	*19.5*	*100.0*	*59.3*	*74.4*	*55.3*
MiniCache	31.1	38.5	41.4	54.7	50.0	36.9	28.0	23.9	23.7	67.5	87.5	41.7	23.0	82.3	32.4	50.9	44.6
MiniCache-V	32.9	42.2	53.8	64.6	56.4	42.2	19.9	23.1	21.6	77.0	93.5	45.5	4.6	100.0	52.8	66.8	49.8
CommonKV	37.4	50.2	54.6	68.8	66.3	50.4	31.3	24.7	24.2	77.0	92.2	49.4	20.5	100.0	62.3	73.3	**55.2**
VFOLD-NAIVE	36.6	47.6	53.5	67.7	65.7	49.6	25.2	23.6	21.1	73.5	93.8	46.1	19.5	100.0	58.1	71.7	53.3
VFOLD	36.6	50.3	55.6	68.5	64.9	49.8	30.8	24.4	23.6	76.5	93.4	49.2	18.5	100.0	59.8	73.5	54.7
Qwen3-8B																	
Full KV	*27.0*	*48.3*	*53.9*	*59.5*	*43.0*	*36.3*	*33.5*	*24.0*	*24.8*	*71.5*	*90.7*	*44.2*	*1.0*	*100.0*	*69.1*	*65.6*	*49.5*
MiniCache	4.7	10.9	14.5	10.1	4.9	4.1	10.9	16.5	13.3	44.0	59.4	20.0	1.0	11.2	32.9	30.2	18.0
MiniCache-V	19.5	32.3	42.8	40.7	26.2	23.9	24.3	21.0	21.1	70.0	90.0	36.1	0.0	95.3	59.0	50.8	40.8
CommonKV	27.5	43.3	45.6	52.9	41.1	29.7	31.6	22.3	23.5	69.5	89.0	40.9	14.0	100.0	68.8	62.3	47.6
VFOLD-NAIVE	25.3	42.2	47.9	58.5	41.7	35.5	25.2	22.0	20.3	67.0	90.5	40.9	2.0	99.0	67.3	62.2	46.7
VFOLD	27.5	46.0	52.2	59.2	41.7	37.5	31.6	23.3	23.5	71.0	90.7	42.4	2.0	100.0	68.8	65.2	**48.9**

tion separately from cache items (Liu et al., 2024a). By default, caches from layers $N_{\text{layers}}/2$ to N_{layers} are merged, resulting in a 25% total cache reduction. We evaluate both the original *MiniCache* and *MiniCache-V*, which is our modified baseline that applies the same compression across all layers, but exclusively to the value cache to match the same 25% total reduction. We adopt the default interpolation hyperparameter $t = 0.6$, and set the token retention threshold to $\gamma = 0.02$.

We additionally include CommonKV, which projects keys and values into a shared low-rank subspace via SVD before cross-layer cache merging (Wang et al., 2025). While related to xKV (Chang et al., 2026), CommonKV avoids xKV's per-request SVD overhead during prefill, which is costly and less practical for batched requests, by using a single subspace across requests. Because CommonKV does not natively support QK-Normalization, present in Qwen3-8B, we adapt the method by applying the QK-Norm after reconstructing keys from compressed latents. Finally, CommonKV is a prefill-only compression method, but is otherwise closely related to our method. We adopt the default CommonKV group size of 4, and fix the number of groups and rank r per layer uniquely per model to match our target compression ratio exactly, remaining close to the default $r = 0.7$. Exact hyperparameters per setting are in Appendix E.

VFOLD Settings We evaluate two variants of our method. VFOLD includes folded alignment maps and cache averaging. VFOLD-NAIVE applies only cache averaging. Comparing these isolates the effect of alignment maps and both demonstrate the utility of post-attention merging. For alignment map learning, we calibrate on 128 Wikitext-2 samples with length 2048 tokens. Following prior work (Xiao et al., 2024; Jiang et al., 2024; Liu et al., 2024b; Staniszewski & Lancucki, 2026), we protect 4 attention sinks and a sliding window of 128 recent tokens. Tokens are merged and added to the shared cache as they leave this window.

6 RESULTS

6.1 PERFORMANCE ON LONG-CONTEXT BENCHMARKS

Tables 2 and 3 report RULER and LongBench results at a 50% value cache reduction (25% of the total KV cache). VFOLD retains at least 98.3% of full-cache performance on every model and

benchmark. It also achieves the best performance compared to baselines in five of six settings; the exception is Mistral-Small-3.1-24B-Instruct on LongBench, where both CommonKV and VFOLD perform similarly to the uncompressed baseline (within 1.2% of the full cache). VFOLD outperforms CommonKV more substantially on RULER for Llama-3.1-8B (+14.2pp) and Qwen3-8B (+2.8pp).

The MiniCache and MiniCache-V results support the key-value asymmetry claims in Section 3. Applying MiniCache to values (MiniCache-V) substantially improves performance at the same compression ratio. VFOLD still outperforms MiniCache-V by a substantial margin on both RULER and LongBench, which demonstrates the importance of design choices in VFOLD, including alignment and compressing the current value vector after attention.

Comparing VFOLD with VFOLD-NAIVE isolates the contribution of the folded alignment maps. This alignment shows improvements in overall RULER and LongBench scores, with RULER scores improving by over 9 percentage points on average. On select tasks, like CWE, alignment recovers a substantial amount of performance lost to direct averaging, like 2.2→75.3 in the case of Llama-3.1-8B-Instruct. This indicates that the alignment is providing substantive value in cache merging, modeling similarities between caches that can then be exploited during subsequent cache averaging. At greater compression (Appendix D), alignment recovers even more performance over VFOLD-NAIVE. However, even without alignment, VFOLD-NAIVE demonstrates the utility of cache averaging; for Llama-3.1-8B-Instruct, this simplified strategy even surpasses CommonKV on RULER.

Table 4: KV efficiency measurements of Llama-3.1-8B-Instruct at 8192-token prompt with 256 token generation (NVIDIA A100-80G, batch size 1 unless specified, 50% value reduction). KV memory is measured after generation and includes protected tokens. Max batch is the largest batch that fits a fixed 80 GB memory budget. Timing experiments are averaged over 3 runs.

	Memory Profile		Performance		
Method	KV red.	KV mem. (GB) ↓	TTFT (ms) ↓	TPOT (ms) ↓	Max batch ↑
Full KV	*0%*	*1.11*	*781*	*21.0*	*33*
MiniCache-V	25%	0.84	794	34.3	**38**
CommonKV	25%	0.86	885	84.7	34
VFOLD	25%	0.84	**786**	**27.8**	**38**

6.2 EFFICIENCY BENCHMARKS

Many cross-layer KV cache compression methods save decoding memory at the cost of extra computation during prefill, extra per-step computations, or both. Table 4 reports KV memory, time-to-first-token (TTFT), time-per-output-token (TPOT), and the largest batch that fits an A100-80G GPU budget for Llama-3.1-8B-Instruct at 8K context and 256 generated tokens (see Appendix F for additional details). We measure VFOLD, MiniCache-V, CommonKV, and a FullKV baseline.

All three compressed methods achieve close to the theoretical 25% memory reduction; the small discrepancies are due to protected tokens, and CommonKV's larger discrepancy arises because generated tokens remain uncompressed. Regarding TTFT, VFOLD leaves prefill timing essentially unchanged (781 vs. 786 ms), since its alignment maps are folded into the weights offline, whereas CommonKV adds 104 ms (+13%). During decoding, VFOLD adds the least overhead of the three compression methods. MiniCache-V restores merged vectors with their separately-stored norm at each decoding step, and CommonKV expensively reconstructs full keys and values from their low rank stored version at each step. VFOLD's remaining 6.8 ms over the full cache likely comes from maintaining the sliding buffer of protected tokens; attention itself runs unmodified. The memory savings are also observed in the maximum attainable batch sizes; VFOLD fits 38 sequences versus 33 for the full cache (+15%), while CommonKV fits only 34. CommonKV's reconstruction step allows for the actual KV memory to remain compressed, but while decoding, all full KV vectors are eventually materialized during the up-projection, resulting in worse memory use.

6.3 COMPOSITION WITH OTHER COMPRESSION METHODS

VFOLD compresses only the value cache and only along the depth dimension, so it should compose well with methods that compress keys or reduce precision. We test two such compositions.

Table 5: Performance of VFOLD and ThinK applied together, as compared to an uncompressed cache baseline, as well as the individual methods.

		Llama-3.1-8B-Inst.		Mistral-Small-3.1-24B-Inst.		Qwen3-8B	
Method	KV red.	LongBench	RULER	LongBench	RULER	LongBench	RULER
Full cache	0%	50.04	92.48	55.33	95.94	49.52	93.14
ThinK	25%	49.65	88.90	52.53	92.44	49.28	93.05
VFOLD	25%	49.40	91.14	54.72	95.12	48.92	91.63
ThinK+VFOLD	50%	48.83	86.32	52.85	87.32	49.00	91.19

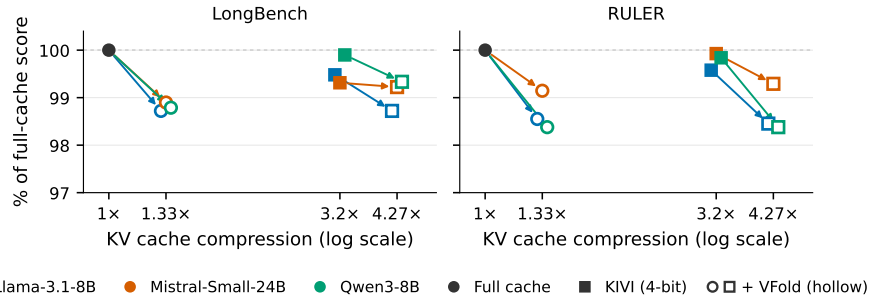


Figure 5: VFOLD composed with 4-bit KIVI. Each arrow adds VFOLD to a base configuration (filled marker) and ends at the combined configuration (hollow marker). Every arrow is the cost of adding VFOLD. Scores are relative to the full cache. Adding VFOLD to KIVI is roughly equivalent to adding VFOLD alone, allowing for greatly improved compression ratios.

Key cache pruning. ThinK prunes key channels using query-driven importance scores (Xu et al., 2025). We prune half of the key channels (25% total KV reduction) and add VFOLD to reach 50% cache compression, a ratio that neither key-only nor value-only pruning can reach since it would require removing an entire cache. Table 5 demonstrates that the two methods pair well to achieve 50% cache compression, without any detrimental performance drop in their composition. Performance lost when combining the two methods is roughly additive, with the slight exception of Mistral on RULER, indicating that they act largely orthogonally and can compose well for greater compression.

Quantization. KIVI quantizes keys per channel and values per token (Liu et al., 2024b); here, we apply VFOLD and then apply 4-bit KIVI. Figure 5 shows that adding VFOLD to KIVI improves compression from $3.2\times$ to $4.27\times$ at almost exactly the performance of VFOLD alone: KIVI+VFOLD is within 0.3 points of VFOLD on every model and benchmark, and even slightly outperforms it in three of six cases. Quantization and cross-layer merging errors do not compound, consistent with these two methods acting on orthogonal axes. Additionally, since VFOLD recovers full value vectors, KIVI applies without modification, which simplifies this composition in practice.

7 CONCLUSION

We introduce VFOLD, a low-overhead value cache compression method that exploits weight symmetries in attention to fold low-cost alignment maps into models in order to later combine caches for memory savings and negligible performance degradation. VFOLD is motivated by several observations, including high inter-layer value cache similarity, high tolerance to cross-layer merging, and an attention-space symmetry unique to the value cache. We exploit this symmetry to fold per-head alignment maps into value and output projections to expose cross-layer redundancy while leaving the model's function unchanged; this allows decoding to run on unmodified attention, solely modifying the cache. We find that VFOLD outperforms similar cross-layer cache reduction methods and an unaligned baseline in both accuracy and efficiency metrics, demonstrating its performance capabilities and the utility of the alignment step. Because VFOLD applies to the depth dimension and specifically on the value cache, it composes well with quantization and key cache-only methods.

AI USE STATEMENT

In this work, we used generative AI tools for providing feedback on research methodology and experiments, implementing methods, supporting qualitative and thematic data analysis, and interpreting results. We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, proposing or refining hypotheses, formulating mathematical claims, assisting in the writing of proofs, or assisting with translation. Generating synthetic datasets, cleaning and reformatting datasets, providing critical ingredients for proving mathematical claims, are not applicable to this work. Additionally, we used generative AI tools for creating scientific figures, creating and editing software code, identifying relevant literature, and editing a research paper for improving readability. We have reviewed all AI-assisted work. We verify LLM-generated code and test it for correctness. We additionally write the full draft and use generative AI only to aid in structuring and writing improvements. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*, 2023.

Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. L-eval: Instituting standardized evaluation for long context language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Saleh Ashkboos, Maximilian Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. SliceGPT: Compress large language models by deleting rows and columns. In *International Conference on Learning Representations*, 2024a.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 2024b.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D. C., Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B. Ashok, and Shashank Shet. Codeplan: Repository-level coding using LLMs and planning. *Proc. ACM Softw. Eng.*, 2024.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. *Advances in Neural Information Processing Systems*, 2024.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed Abdelfattah, and Kai-Chiang Wu. Palu: Kv-cache compression with low-rank projection. In *International Conference on Learning Representations*, 2025.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Hung-Yueh Chiang, Yash Akhauri, Xilai Dai, Huiqiang Jiang, Yucheng Li, Luis Ceze, Kai-Chiang Wu, and Mohamed S. Abdelfattah. xKV: Cross-layer KV-cache compression via aligned singular vector extraction. In *Forty-third International Conference on Machine Learning*, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. A simple and effective l_2 norm-based strategy for KV cache compression. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020.

Stefan Horoi, Albert Manuel Orozco Camacho, Eugene Belilovsky, and Guy Wolf. Harmony in diversity: Merging neural networks with canonical correlation analysis. In *Proceedings of the 41st International Conference on Machine Learning*, Proceedings of Machine Learning Research, pp. 18815–18832. PMLR, 21–27 Jul 2024.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.

Shiyu Ji, Yixuan Wang, Yijun Liu, Qingfu Zhu, and Wanxiang Che. EchoKV: Efficient KV cache compression via similarity-based reconstruction. *arXiv preprint arXiv:2603.22910*, 2026.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

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. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, 2019.

Hongzhan Lin, Zhiqi Bai, Xinmiao Zhang, Sen Yang, Xiang Li, Siran Yang, Yunlong Xu, Jiaheng Liu, Yongchi Zhao, Jiamang Wang, et al. Reconstructing KV caches with cross-layer fusion for enhanced transformers. *International Conference on Learning Representations*, 2026.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. MiniCache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 2024a.

Alexander H Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. SpinQuant: Llm quantization with learned rotations. In *International Conference on Learning Representations*, 2025.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: a tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, 2024b.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems*, 2023.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for KV cache compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15332–15344, 2024.

Alina Shutova, Vladimir Malinovskii, Vage Egiazarian, Denis Kuznedelev, Denis Mazur, Surkov Nikita, Ivan Ermakov, and Dan Alistarh. Cache me if you must: Adaptive key-value quantization for large language models. In *International Conference on Machine Learning*, pp. 55451–55473. PMLR, 2025.

Konrad Staniszewski and Adrian Lancucki. KV cache transform coding for compact storage in LLM inference. In *International Conference on Learning Representations*, volume 2026, pp. 37768–37815, 2026.

George Stoica, Daniel Bolya, Jakob Bjorner, Pratik Ramesh, Taylor Hearn, and Judy Hoffman. Zipit! merging models from different tasks without training. In *International Conference on Learning Representations*, volume 2024, pp. 29215–29237, 2024.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Zunhai Su, Hanyu Wei, Zhe Chen, Wang Shen, Linge Li, Huangqi Yu, and Kehong Yuan. RotateKV: accurate and robust 2-bit KV cache quantization for LLMs via outlier-aware adaptive rotations. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, 2025.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *Advances in Neural Information Processing Systems*, 37:7339–7361, 2024.

Qian Tao, Wenyuan Yu, and Jingren Zhou. AsymKV: Enabling 1-bit quantization of KV cache with layer-wise asymmetric quantization configurations. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 2316–2328, January 2025.

Neha Verma and Maha Elbayad. Merging text transformer models from different initializations. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

Yixuan Wang, Haoyu Qiao, Lujun Li, Qingfu Zhu, and Wanxiang Che. CommonKV: Compressing KV cache with cross-layer parameter sharing. *arXiv preprint arXiv:2508.16134*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Haoyi Wu and Kewei Tu. Layer-condensed KV cache for efficient inference of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11175–11188, 2024.

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, pp. 21875–21895, 2024.

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. ThinK: Thinner key cache by query-driven pruning. In *International Conference on Learning Representations*, volume 2025, pp. 56691–56709, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yifei Yang, Zouying Cao, Qiguang Chen, Libo Qin, Dongjie Yang, Hai Zhao, and Zhi Chen. KVSharer: Efficient inference via layer-wise dissimilar kv cache sharing. *arXiv preprint arXiv:2410.18517*, 2024.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. TurboQuant: Online vector quantization with near-optimal distortion rate, 2025.

Binchi Zhang, Zaiyi Zheng, Zhengzhang Chen, and Jundong Li. Beyond the permutation symmetry of transformers: The role of rotation for model fusion. In *Forty-second International Conference on Machine Learning*, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2O: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 2023.

A VFOLD MAP DETAILS FOR GROUPED QUERY ATTENTION

All evaluated models in this work include Grouped Query Attention (GQA) with n_{q} query heads and n_{kv} KV heads, of dimension d_h and group size $g = n_{\text{q}}/n_{\text{kv}}$. As for weights, we have $W_Q \in \mathbb{R}^{d_{\text{model}} \times n_{\text{q}} d_h}$, $W_K, W_V \in \mathbb{R}^{d_{\text{model}} \times n_{\text{kv}} d_h}$, and $W_O \in \mathbb{R}^{n_{\text{q}} d_h \times d_{\text{model}}}$.

The head assignment problem in Equation (4) only applies to n_{kv} heads, since this is how value caches are stored in GQA. Let $P_{\pi(i),i} = 1$ so that new KV head i is original KV head $\pi(i)$. Permuting a KV head requires moving all g associated query heads with it, so we expand P via Kronecker product at the appropriate block size:

$$P_{\text{kv}} = P \otimes I_{d_h} \qquad P_{\text{q}} = P \otimes I_{g d_h} \tag{5}$$

The Kronecker product allows each block of query heads to move as a unit, in order to maintain their alignment with the corresponding KV heads. Similarly, for W_O, T must be expanded to the full model dimension from the GQA dimension. Therefore, we define $T_q = \text{blockdiag}(I_g \otimes T_1, \ldots, I_g \otimes T_{n_{\text{kv}}})$, which repeats each KV head's map T for each of its g query heads.

For the full equations of the attention weights that receive folded maps, we have the following:

$$W_Q \leftarrow W_Q P_{\text{q}} \qquad W_K \leftarrow W_K P_{\text{kv}} \qquad W_V \leftarrow W_V P_{\text{kv}} T \qquad W_O \leftarrow T_{\text{q}}^{-1} P_{\text{q}}^T W_O \tag{6}$$

B CCA AND ITS INVERTIBILITY

We review details on Canonical Correlation Analysis to describe how we compute the alignment maps. For each pair of matched KV heads, we find a full-rank $d_h \times d_h$ map using CCA. For value caches $V_\ell^h, V_{\ell+1}^h \in \mathbb{R}^{n \times d_h}$ over n calibration tokens, we define the following products. For the numerical stability of the inverse square root, we set $\lambda = 10^{-4}$.

$$\Sigma_{11} = \frac{1}{n}(V_\ell^h)^T V_\ell^h + \lambda I \qquad \Sigma_{22} = \frac{1}{n}(V_{\ell+1}^h)^T V_{\ell+1}^h + \lambda I \qquad \Sigma_{12} = \frac{1}{n}(V_\ell^h)^T V_{\ell+1}^h \tag{7}$$

We then whiten Σ_{12} as follows, and take its SVD:

$$USV^T = \Sigma_{11}^{-1/2} \Sigma_{12} \Sigma_{22}^{-1/2} \tag{8}$$

The alignment matrix is the following, after obtaining the SVD:

$$T = \Sigma_{22}^{-\frac{1}{2}} V U^T \Sigma_{11}^{\frac{1}{2}} \tag{9}$$

In order to maintain functional equivalence using the weight symmetry, we must compute the inverse of T for our method. In our implementation, we use the numerical inverse. However, we also note that the inverse of this map is expressible as follows:

$$T^{-1} = \Sigma_{11}^{-\frac{1}{2}} U V^T \Sigma_{22}^{\frac{1}{2}} \tag{10}$$

T is the map from Section 4 ($U_{\ell+1} U_l^{-1}$).

The invertibility of these maps is of interest to determine whether this is a source of error due to numerical stability. To investigate this, we report the condition number and numerical error of maps used for Llama-3.1-8B-Instruct, for 2-cache groups. For error, we fold the maps, and then compare (L2 error) the output of the value-output product to the original value-output product.

Table 6: Stability of the CCA maps for Llama-3.1-8B-Instruct, over all matched head pairs. κ is the condition number. Inverse error compares float32 inverse used when folding with the closed form. Value-output distance is the measure of the distance between $W_V^h W_O^h$ products, with and without mapping.

	Average	Max
$\kappa(T)$	6.4	42.1
Inverse error (fp32)	1.4×10^{-6}	1.6×10^{-6}
VO-distance (bf16)	0.28%	0.71%

The results from the table demonstrate that 1) the maps are well-conditioned, so their numerical inverse is stable and close to the exact one, and 2) that the distances between value-output products, before and after folding, are negligible.

C Cache similarities on wikitext

We additionally measure the cache similarities on calibration data for the various value cache representations as described in Section 4. In Figure 6, we report cosine similarities between caches, and find similar patterns to those found in Figure 3, with average similarity between aligned caches measuring closer to 0.9 rather than 0.85 as was the case on held-out data.

This suggests that the data used to fit the maps, Wikitext in this case, is general enough to suffice in creating the alignment maps, and/or that the alignment maps are not terribly sensitive to the distribution shift between alignment and test data (GSM8K in Section 4).

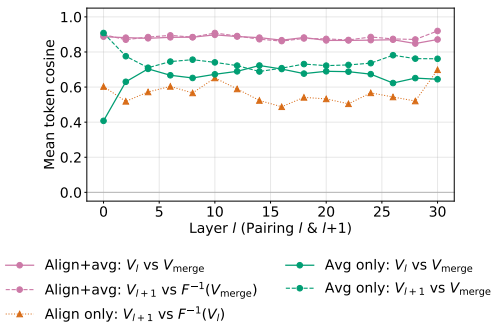


Figure 6: Similarity between combined caches and original value caches. We construct the aligned average as $V_{\text{merge}} = \frac{1}{2}(V_l + F(V_{\ell+1}))$ and compare it with V_l, for both $F = I$ (unaligned) and CCA. We also map the merged cache back to the coordinate system of layer $\ell + 1$ and compare $F^{-1}(V_{\text{merge}})$ with $V_{\ell+1}$. The aligned average achieves cosine similarities close to 0.9 on in-distribution data.

D Extending VFold to more layers

Extending VFold The pairwise construction of VFold ($k = 2$) can generalize to groups of k consecutive layers, which reduces the value cache to $1/k$ of its original size. In this section, we discuss this extension, and then results for a 75% value cache reduction (37.5% total KV reduction) for $k = 4$.

Given k and a starting layer ℓ, we designate a middle layer $\ell + \lfloor k/2 \rfloor$[6] from the group as our reference, and align every other layer $\ell + j$, $(j = 0, \ldots, k-1)$ directly to it: we compute head-level CCA costs between layers ℓ and $\ell + j$, solve the head assignment with the Hungarian algorithm, and fold

[6]Pairs are a special case, where we use ℓ rather than $\ell + 1$ as the reference.

the resulting permutation P_j and block-diagonal map T_j into layer $\ell + j$ exactly as in the pairwise case, with T_j^{-1} and $P_j^\top$ folded into W_O. Because each map acts on a different layer's weights, the $k - 1$ alignments are independent and the reparameterized model remains exactly function-preserving. The shared cache stores

$$V_{\text{merge}} = \frac{1}{k}\Big(\sum_{j=0}^{k-1} F_j(V_{\ell+j})\Big), \qquad F_j = I \text{ if } j = \ell + \lfloor k/2 \rfloor, \text{else } F_j = P_j T_j \tag{11}$$

where F_j denotes the alignment folded into layer $\ell + j$. Aligning to a common reference, rather than chaining pairwise maps, avoids compounding alignment error across the group. Calibration data and protected tokens are identical to the $k = 2$ setting.

Table 7: Performance comparisons of KV cache compression methods on LongBench. All results are at 37.5% reduction, matching the 75% value cache reduction for VFOLD. We bold the best results, and underline the second-best.

Methods	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot			Synth.		Code		
	Nrtv	Qasp	MF-en	Hotpot	2Wiki	Musiq	Gov	QMS	MN	TREC	TQA	SAM	PC	PRe	LCC	RB-P	Avg
Llama-3.1-8B-Instruct																	
Full KV	*31.3*	*46.8*	*56.6*	*58.1*	*48.9*	*32.5*	*34.5*	*25.3*	*27.0*	*72.5*	*91.7*	*43.7*	*8.6*	*100.0*	*65.1*	*58.3*	*50.0*
MiniCache	6.3	3.7	11.2	9.5	4.1	4.0	4.4	14.1	6.3	44.0	45.9	18.2	0.9	51.4	29.2	30.6	17.7
CommonKV	30.5	41.7	54.7	54.4	46.4	30.3	26.1	24.3	25.0	61.0	90.8	42.3	5.5	99.5	62.5	57.7	<u>47.0</u>
VFOLD-NAIVE	30.8	30.1	44.9	55.5	44.6	29.1	18.0	21.8	18.0	36.0	91.0	38.2	9.3	100.0	57.4	49.4	42.1
VFOLD	33.1	44.1	57.1	58.3	46.2	31.1	26.8	24.7	23.5	71.5	90.4	40.9	6.9	99.5	61.3	55.2	**48.2**
Mistral-Small-3.1-24B-Instruct																	
Full KV	*37.5*	*51.5*	*54.8*	*68.9*	*65.7*	*50.2*	*33.1*	*25.0*	*24.7*	*77.0*	*93.2*	*50.3*	*19.5*	*100.0*	*59.3*	*74.4*	*55.3*
MiniCache	14.2	25.2	27.1	40.2	26.9	20.0	11.3	19.1	13.9	68.0	84.4	31.4	3.5	55.0	25.1	38.3	31.5
CommonKV	36.1	49.5	55.2	69.5	66.1	51.1	30.7	24.5	23.9	77.5	92.5	49.0	19.5	100.0	63.4	73.2	**55.1**
VFOLD-NAIVE	32.8	39.7	46.3	65.1	63.2	46.9	20.3	21.2	15.8	41.0	93.4	44.0	17.0	100.0	54.8	64.6	47.9
VFOLD	36.9	50.0	55.4	68.4	65.5	50.2	28.5	23.7	22.1	77.0	93.6	48.6	18.0	100.0	60.3	71.9	<u>54.4</u>
Qwen3-8B																	
Full KV	*27.0*	*48.3*	*53.9*	*59.5*	*43.0*	*36.3*	*33.5*	*24.0*	*24.8*	*71.5*	*90.7*	*44.2*	*1.0*	*100.0*	*69.1*	*65.6*	*49.5*
MiniCache	0.9	4.3	5.2	2.7	2.8	1.2	4.8	13.4	7.6	30.5	21.1	9.1	0.0	0.5	30.6	24.6	10.0
CommonKV	26.7	43.6	45.5	52.9	40.7	29.0	31.9	22.6	23.6	70.5	88.1	40.0	13.0	99.5	67.9	62.1	<u>47.3</u>
VFOLD-NAIVE	17.9	30.4	36.2	52.4	36.0	26.9	18.7	19.9	14.2	47.5	88.2	35.6	3.0	96.8	62.7	52.9	40.0
VFOLD	27.8	45.5	50.5	60.1	43.1	35.7	27.7	22.7	22.0	71.0	91.2	41.3	4.0	100.0	67.8	64.0	**48.4**

Baselines At this new compression ratio, we remove MiniCache-V as a baseline because the method uses SLERP to combine caches, and is not specified for groups larger than 2. For MiniCache, we apply the method to $N_{\text{layers}}/4$ until N_{layers}. For Qwen, we start at layer 8 instead of layer 9 since we need to group layers by 2. CommonKV uses the 75% settings in Table 9.

Merging Execution Schedule In order to avoid storing k layers' caches simultaneously during execution before merging, we adopt an incremental averaging approach by combining 2 layers at a time, meaning only one additional cache is ever in buffer during averaging, which reduces peak activation memory during inference.

For the sake of simplicity, we assume all caches are pre-aligned, and write each as V_i. To accommodate the average in Equation (11), we write the incremental mean as the following. We utilize relative indexing with 0 representing the first cache in the group. Let $\hat{V}_i$ represent the ith buffer. We initialize the buffer with the first cache, $\hat{V}_0 = V_0$.

$$\hat{V}_i = \alpha_i \hat{V}_{i-1} + (1 - \alpha_i) V_i, \qquad \alpha_i = \frac{i}{i+1} \tag{12}$$

Briefly, we demonstrate this incremental mean gives the average from Equation (11).

Claim. Given Equation (12), $\hat{V}_k = \frac{1}{k+1} \sum_{i=0}^{k} V_i$ for all k.

Table 8: Performance comparisons of KV cache compression methods on RULER, at 16K length. All results are at 37.5% reduction, matching the 75% value cache reduction for VFOLD. We bold the best results, and underline the second-best.

	Needle In A Haystack (NIAH)								Synthetic			QA		
Method	**S1**	**S2**	**S3**	**MK1**	**MK2**	**MK3**	**MQ**	**MV**	**CWE**	**FWE**	**VT**	**QA1**	**QA2**	**Avg**
Llama-3.1-8B-Instruct														
Full KV	*100.0*	*100.0*	*100.0*	*99.80*	*100.0*	*99.60*	*99.25*	*98.75*	*81.76*	*91.13*	*98.96*	*79.40*	*53.60*	*92.48*
MiniCache	3.60	0.40	0.00	0.40	0.20	0.00	0.00	0.00	0.10	8.00	6.00	21.20	15.80	4.28
CommonKV	96.80	73.60	63.00	83.40	62.00	4.00	76.55	62.65	14.48	88.80	80.04	73.40	49.60	63.72
VFOLD-NAIVE	0.20	19.60	16.00	19.40	30.00	0.40	3.35	2.65	0.04	3.87	16.44	67.20	45.60	17.29
VFOLD	99.20	90.80	92.80	94.00	94.40	89.40	93.55	84.35	41.20	84.07	91.12	77.00	50.80	83.28
Mistral-Small-3.1-24B-Instruct														
Full KV	*100.0*	*99.80*	*100.0*	*99.80*	*100.0*	*99.40*	*100.0*	*99.95*	*96.48*	*98.73*	*100.0*	*85.00*	*68.00*	*95.94*
MiniCache	29.80	7.00	0.00	2.80	0.00	0.00	0.90	0.80	4.10	50.30	36.20	41.60	32.60	15.85
CommonKV	100.00	99.80	100.00	100.00	99.80	99.40	99.90	99.95	90.28	94.73	100.00	82.80	64.60	94.71
VFOLD-NAIVE	84.80	81.80	24.00	86.40	79.00	5.00	84.50	50.90	15.82	32.33	57.68	76.40	60.40	56.85
VFOLD	100.00	99.60	100.00	99.60	100.00	99.20	99.70	96.45	83.26	97.80	99.68	82.60	65.20	94.08
Qwen3-8B														
Full KV	*100.0*	*100.0*	*99.80*	*99.80*	*100.0*	*99.80*	*99.85*	*99.55*	*83.76*	*93.80*	*100.0*	*75.40*	*59.00*	*93.14*
MiniCache	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50	15.30	0.00	6.60	8.00	2.39
CommonKV	100.00	100.00	98.60	98.40	98.40	74.20	99.60	97.00	57.78	93.33	99.04	66.20	54.40	**87.46**
VFOLD-NAIVE	53.20	25.60	3.40	24.40	0.80	0.00	5.85	5.55	4.02	0.13	8.76	58.80	44.80	18.10
VFOLD	99.80	100.00	98.20	99.00	99.40	41.20	99.65	99.30	57.58	86.47	99.08	72.20	55.40	<u>85.18</u>

Proof. We prove the claim by induction. The base case holds trivially, $\hat{V}_0 = V_0$.

Now, assume for induction that $\hat{V}_{k-1} = \frac{1}{k}\sum_{i=0}^{k-1} V_i$. Then,

$$\hat{V}_k = \frac{k}{k+1} \cdot \frac{1}{k} \sum_{i=0}^{k-1} V_i + \frac{1}{k+1} V_k = \frac{1}{k+1} \sum_{i=0}^{k} V_i \tag{13}$$

□

Results Table 7 reports LongBench results for $k = 4$, and Table 8 reports RULER. On LongBench, VFOLD retains 96.4%, 98.4%, and 97.8% of full-cache performance on Llama, Mistral, and Qwen, outperforming CommonKV on Llama (+1.2) and Qwen (+1.1) and trailing it slightly on Mistral (-0.7). RULER demonstrates greater performance spread at this ratio; VFOLD outperforms CommonKV by a wide margin on Llama (83.3 vs. 63.7), where CommonKV fails on the hardest multi-key task (MK3: 4.0 vs. 89.4). CommonKV is ahead on Mistral (-0.6), mostly due to CWE, and on Qwen (-2.3), where the gap comes almost entirely from MK3 (41.2 vs. 74.2) and FWE. On the remaining eleven tasks, VFOLD is within 0.4 points of CommonKV or ahead.

The value of alignment grows markedly with group size. On LongBench, the gap between VFOLD and VFOLD-NAIVE widens from $\approx$ 2 points at $k = 2$ to $\approx$ 7 points at $k = 4$. On RULER this discrepancy widens further, ranging from between 4.0–12.8 points at $k = 2$ to between 37.2–67.1 points at $k = 4$. Unaligned averaging (VFOLD-NAIVE) of four caches collapses on RULER (17.3, 56.9, and 18.1 on Llama, Mistral, and Qwen), while aligned averaging does not. MiniCache degrades severely at this ratio on both benchmarks.

E COMMONKV SETTINGS

We report exact CommonKV group and rank ratio hyperparameter settings for the three models we test, as well as the two values compression ratios, found in Table 9. For all CommonKV experiments, the first 4 sink tokens and 128 recent tokens (non-sliding, as CommonKV compresses prefill only) are not merged as per the proposed method. Finally, we adopt the paper's Fisher warmup settings of 2048 samples of length 1024 sequences of Wikitext2.

Table 9: Hyperparameter settings for CommonKV on each model. Group size is fixed at 4 for all experiments. We select the number of groups to merge while keeping the rank around 0.7 as per the original paper, but allow the rank to vary to accommodate the exact compression ratio.

Values Ratio	Model	Layers	Merged Group Fraction	Rank
50%	Llama-3.1-8B-Instruct	32	5/8	0.7008
	Qwen3-8B	36	6/9	0.7440
	Mistral-Small-3.1-24B-Inst.	40	8/10	0.7411
75%	Llama-3.1-8B-Instruct	32	6/8	0.7069
	Qwen3-8B	36	7/9	0.7416
	Mistral-Small-3.1-24B-Inst.	40	9/10	0.7565

F TIMING EXPERIMENTS DETAILS

All efficiency measurements are computed using a single NVIDIA A100-80GB GPU with PyTorch 2.5.1, CUDA 12.1, in float16. Attention is implemented using SDPA. We build the length 8192 prompts by sampling token IDs uniformly at random over the vocabulary and greedy decoding for 256 tokens.

TPOT is the mean over the 255 decoding steps, because the first decoding step is computed from prefill. TTFT is the total time from the start of prefill until the sampling of the first token. We use 2 warm-up and 3 timed trials, and report the mean.