KV-Cache Transform Coding for Compact Storage in LLM Inference

Anonymous authors
Paper under double-blind review

Abstract

Serving large language models (LLMs) at scale necessitates efficient key-value (KV) cache management. KV caches can be reused across conversation turns via shared-prefix prompts that are common in iterative code editing and chat. However, stale caches consume scarce GPU memory, require offloading, or force recomputation. We present `kvtc`, a lightweight transform coder that compresses KV caches for compact on-GPU and off-GPU storage. Drawing on classical media compression, `kvtc` combines PCA-based feature decorrelation, adaptive quantization, and entropy coding. It requires only a brief initial calibration and leaves model parameters unchanged. By exploiting redundancies in KV caches, `kvtc` achieves up to 20x compression, while maintaining reasoning and long-context accuracy in Llama 3.1, Mistral-NeMo, and R1-Qwen 2.5. Across benchmarks including AIME25, LiveCodeBench, GSM8K, MMLU, Qasper, RULER, and MATH500, `kvtc` consistently outperforms inference-time baselines such as token eviction, quantization, and SVD-based methods, delivering substantially higher compression ratios. These results support `kvtc` as a practical building block for memory-efficient LLM serving with reusable KV caches.

1 Introduction

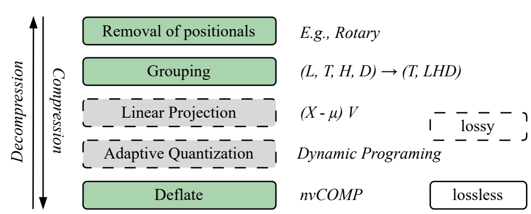
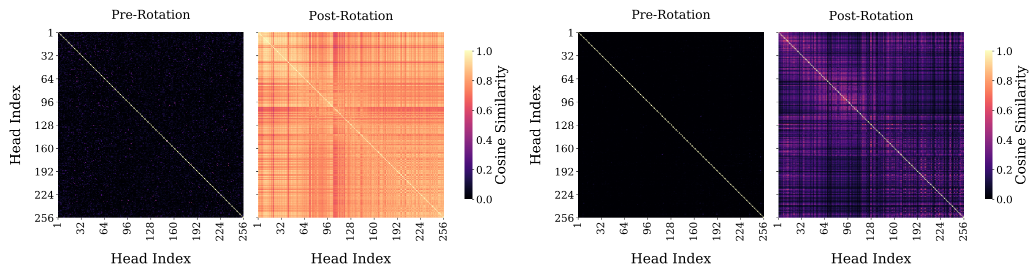


Figure 1: The `kvtc` transform-coding pipeline is applied separately to the key and value caches. Here, T, L, H, and D denote the time, layer, head, and head dimension, respectively. Features are linearly decorrelated via PCA, and the resulting PCA coefficients are quantized using variable bit widths. The PCA basis V is computed once on a calibration dataset and reused for all caches.

Chat-based interfaces, commonly used for interacting with large language models (LLMs), enable users to iteratively refine answers across open-domain dialogues and specialized tasks, such as code generation (Chiang et al., 2024; Köpf et al., 2023). Each conversational turn extends the key–value (KV) cache associated with a conversation, storing hidden activations for every previous token. For modern Transformer models, this cache can easily occupy multiple gigabytes. As models scale up in size and reasoning capability, generating increasingly long reasoning chains (OpenAI et al., 2024), the KV cache footprint increases, posing a significant bottleneck for throughput and latency. During user turns, stale KV caches left on-chip occupy memory, which is necessary for serving other users, yet ensures the fastest responses in the future. Conversely, caches could be discarded, incurring the cost of recomputation, or offloaded to CPU DRAM or local/network storage, leading to transfer overheads. This tension creates a latency–throughput dilemma in production systems and necessitates careful configuration.

Crucially, inference frameworks view the local KV caches as databases and strategies like block-level paging and prefix sharing promote reuse of caches whenever prompt prefix matches (Kwon et al., 2023). Scaling LLM serving increasingly hinges on KV cache management and reuse (Liu et al., 2024; Cheng et al., 2024; Yao et al., 2025), but current systems struggle to store, move, and refresh these caches efficiently. CacheGen (Liu et al., 2024) compresses caches for transmission, offering at most 4.3× KV cache reduction in comparison to the FP8 quantization baseline. SVDq



(a) Keys (b) Values

Figure 2: Cosine similarity before and after alignment between key (a) and value (b) heads calculated using Llama 3.1 8B on inputs from Qasper (Dasigi et al., 2021; Shaham et al., 2022). For each example, we calculate cosine similarity between all keys/values from the same position and then average across the batch. Orthonormal alignment matrices were produced using 20 samples from the RedPajama v2 (Weber et al., 2024).

(Yankun et al., 2025) and xKV (Chang et al., 2025) pursue low-rank compression during prefill, but both require calculation of per-prompt SVD. Long and frequently used prompts may mandate investing more compute for offline training of corpus-specific caches (Eyuboglu et al., 2025).

Meanwhile, intensively studied KV cache compression methods, aimed at improving the performance of autoregressive generation, offer interim measures to the cache retention problem (Yuan et al., 2024). Prior work hinges on key observations that KV cache can be quantized (Frantar et al., 2023; Lin et al., 2024), sparsified (Zirui Liu et al., 2023; Hooper et al., 2024; Łańcucki et al., 2025) or approximated (Nawrot et al., 2024); the cache itself is compressible (Yuan et al., 2024), and dimensions of keys and values for separate heads show a high degree of correlation (Chang et al., 2025; Oren et al., 2024; Zhang et al., 2023). For long contexts, these methods offer substantial throughput and latency improvements, by lowering KV cache sizes and thus the memory traffic during next token prediction. However, due to tight latency constraints, often coupled with refraining from modifying weights of the model, these techniques tend to be brittle (Tang et al., 2024), and accuracy degradation prohibits combining methods for compounded benefits. Finally, these methods seldom exploit the strong low-rank structure of KV tensors.

In this paper, we introduce `kvtc`: a simple yet powerful transform coding scheme, compressing KV caches for storage. Inspired by classical image codecs, it applies a learned orthonormal transform followed by channel-wise scalar quantization—dynamically allocating bits—and entropy coding. The resulting bit-stream is on average 20× smaller than the original 16-bit one, while maintaining on par accuracy. The method also exposes a smooth rate–accuracy trade-off, with 40× or higher compression attainable at modest, model- and task-dependent accuracy costs. `kvtc` therefore mitigates to a large extent the problem of KV cache management: lowering the cost of its on-chip retention and the bandwidth required for offloading, without compromising interactive latency.

2 Preliminaries

Table 1: KV cache size in 16 bits for 1K tokens of context.

Model	Size
Qwen 2.5 R1 1.5B	28 MiB
Qwen 2.5 R1 7B	56 MiB
Llama 3.1 8B	128 MiB
Llama 3.3 70B Instruct	320 MiB
Mistral-NeMo 12B	160 MiB
MN-Minitron 8B	160 MiB

Structure of the KV Cache During decoding in autoregressive Transformers with multi-head self-attention, the keys and values produced for each processed token are cached to avoid recomputation. The collection of these tensors is the *KV cache*. For l layers, h heads, head dimension d_{head} and sequence length t, a 16-bit KV cache occupies $(4\,l\,h\,d_{head}\,t)$ bytes.

Motivated by work on cross-layer KV cache sharing and compression (Brandon et al., 2024; Chang et al., 2025), we ask whether keys (and analogously values) from different attention heads lie in a shared latent subspace. As a litmus test for linear alignment, we align per-head caches using orthogonal transformations obtained from the orthogonal Procrustes problem (Gower & Dijksterhuis, 2004), as shown in Figure 2.

Before alignment, inter-head cosine similarity is mostly below 0.2; after alignment, it increases substantially for keys and moderately for values. These results suggest that heads largely differ by near-orthogonal transformations, motivating PCA-based dimensionality reduction. We present additional justification in Appendix C.1.

Sliding Windows and Sink Tokens In transform coding for vision and audio, bits are allocated to transform coefficients so that quantization induces minimal perceptual distortion. By loose analogy, in Transformer LMs, the attention mass allocated to a token can be viewed as a proxy for its importance to downstream predictions. A well-documented recency bias causes recent tokens to attract substantially more attention than distant ones (Jiang et al., 2024). Following Zirui Liu et al. (2023), we therefore keep a sliding window of the most recent w tokens uncompressed (typically $w = 128$).

By contrast, *sink tokens*—the first few positions that consistently accumulate attention—do not follow this recency pattern and remain highly attended irrespective of context length (Xiao et al., 2024). Moreover, these positions occupy a distinct linear subspace from the remaining positions (Figure 4). Consequently, we exclude the first $N = 4$ tokens from compression and report an ablation of this choice in Appendix C.

Multi-Turn Conversations Let a conversation be an ordered sequence

$$\mathcal{C} = \langle (x_0,\, y_0),\, (x_1,\, y_1), \ldots \rangle,$$

where x_t denotes the user (or system) input at turn t and y_t the generated reply. Generation of y_t consists of a prefill pass, which generates the KV cache for all preceding tokens $(x_0, y_0, \ldots, x_t)$, followed by incremental decoding, which produces the tokens of y_t one at a time.

When a new user prompt x_i is received, the existing KV cache can be re-used and only newly added tokens have to be forwarded through the model, reducing computation and time-to-first-token (TTFT). However, if the cache has been deleted, the model must reprocess the entire conversation as a prompt, resulting in quadratic recomputation of attention across the input.

KV Cache Management while Serving Efficient LLM deployments often split prefill and decode onto separate nodes because their performance characteristics differ (Zhong et al., 2024). The prefill node produces the KV cache and sends it to the decode node over a high-speed fabric—ideally using RDMA (e.g., InfiniBand/RoCE)—to minimize latency and CPU overhead. Both nodes can maintain tiered KV-cache pools: on GPU HBM (hot), CPU DRAM (warm), and NVMe/SSD (cold), with an eviction policy such as least recently used (LRU) to keep frequently reused caches resident. For long-term residency, caches can be sent to remote storage. When a node is selected for prefill or decode, its choice can be dictated by already held KV cache. In such setups, KV cache transfers are typically the dominant cross-node traffic (Figure 3).

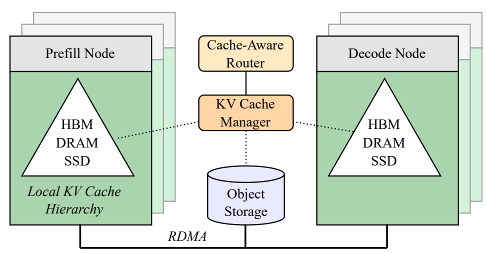


Figure 3: A high-level architecture of KV-cache-aware LLM serving environment.

Why compress KV caches in between prefill/decoding phases? Compression can: ❶ extend the effective capacity and lifetime of local KV stores roughly in proportion to the compression ratio; and ❷ cut network traffic. We discuss both angles below.

❶ Extending KV-cache lifetime in the tiered store raises higher-tier hit rates (HBM/DRAM vs. NVMe) and avoids re-prefill for repeated or overlapping contexts (e.g., code-assist sessions). For example, a single 1,000-line file tokenized to 10 tokens per line and processed by Llama 3.3 70B produces about 1.6 GiB of 8-bit KV cache. On a busy node rotating users—even at moderate batch sizes—KV volume can prevent hot/warm residency. Extending cache lifetime 20× via compression can determine whether a cache remains hot/warm, or must be recomputed.

❷ Prefill time scales as $\mathcal{O}(n^2)$ with prompt length and takes considerably more time than transfer, and the cache can be streamed by layer during prefill (Qin et al., 2025) in order to further reduce TTFT. However, KV cache compression reduces memory traffic proportionally to the compression ratio, which might be critical if the network bandwidth is saturated and becomes a bottleneck.

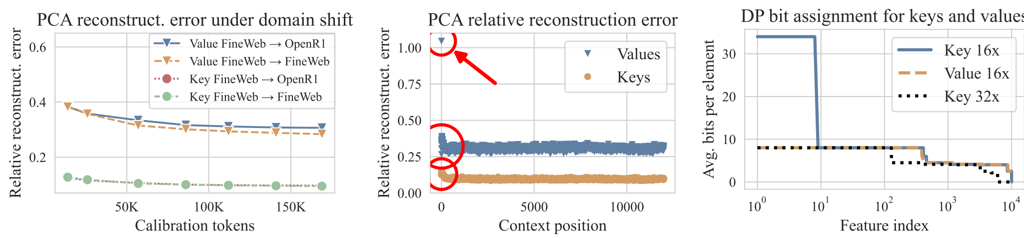


Figure 4: Calibration of Llama 3.1 8B with `kvtc`. **Left:** the reconstruction error decreases as the calibration data size grows. **Middle:** the reconstruction error is higher for the sink tokens. **Right:** Bit assignment computed via dynamic programming, counting in the per-group scaling factors.

3 METHOD

Our Key-Value Transform Coder (`kvtc`) builds upon the transform-coding framework (Ahmed et al., 1974; Goyal, 2001), a widely adopted methodology for designing image and video compression algorithms such as JPEG (Joint Photographic Experts Group, 1994). This framework typically consists of a decorrelation step (commonly implemented via DCT in image/video codecs), followed by quantization and entropy coding.

In our approach, decorrelation is accomplished using an orthonormal projection matrix V derived from the singular value decomposition (SVD) of the centered calibration data (i.e., principal component analysis, PCA). Quantization is optimized via a dynamic programming algorithm, while entropy coding is performed using Deflate (Wu, 2017) (see Figure 1). The `kvtc` algorithm comprises three main phases:

- **Calibration.** During calibration, we collect the key (and similarly value) caches from calibration data. The caches are rearranged by concatenating heads across layers to form the calibration data matrix C, where rows represent tokens and columns correspond to feature dimensions. We center C and store the mean μ used for centering. Next, we compute $\mathrm{SVD}(C - \mu) = U\Sigma V^\top$ and retain V. All calibration steps up to this point are performed once per model, separately for key and value caches. Finally, for a given CR, we use a dynamic programming algorithm to determine the optimal bit allocation (minimizing the Frobenius norm) for each column of $U\Sigma$. Calibration for a 12B model can be completed within 10 minutes on an H100 GPU (Appendix C.5).
- **Compression.** Keys and values are compressed independently using the (V, μ) parameters and bit allocation obtained during calibration. Compression is performed between model phases (e.g., after decoding or between prefill and decoding) and can be executed on either GPU (affecting TTFT) or CPU (if the cache is already in storage). Importantly, during decoding, the model operates on decompressed KV caches; compression is used only for storage or transfer.
- **Decompression.** Decompression reverses the compression steps. The most computationally intensive operation—the inverse projection using $V^\top$—can be performed layer-by-layer using sub-matrices of $V^\top$, allowing generation to begin early.

Below, we provide further details on each component of `kvtc`.

3.1 FEATURE DECORRELATION

Unlike prior SVD-based methods that calculate a separate decomposition for each prompt (Yankun et al., 2025; Chang et al., 2025), we compute the KV cache projection matrices **once**—using a calibration dataset $\mathcal{C}$—and reuse them across all requests at inference time. Preparing a single, generalizable V rests on three observations. First, SVD must be computed on a large, representative sample; sampling token positions from a diverse calibration set suffices for generalization and is computationally tractable. Second, excluding highly attended tokens—the most recent ones and attention sinks—improves the achievable compression ratio (see Tables 6 and 10 in Appendix C). Third, positional embeddings—e.g., Rotary (Su et al., 2023)—distort the apparent low-rank structure of keys and should be removed before compression (Sun et al., 2025).

Computation We forward all sequences from $\mathcal{C}$ through the model and collect their KV caches. For each sequence, cache entries are concatenated along the time dimension to form a global pool of positions. From this pool, we sample n token positions, excluding attention sinks. For each sampled position, we take the corresponding keys (and, equivalently, values) from l layers and h heads, undo positional rotations, and concatenate them along the hidden dimension d_{head}. This yields a data matrix $C \in \mathbb{R}^{n \times p}$ with $p = l\,h\,d_{\text{head}}$, whose rows index sampled token positions. Let $\mu \in \mathbb{R}^p$ be the per-feature mean of C. We compute the SVD of the centered matrix,

$$C - \mu = U\,\Sigma\,V^\top, \tag{1}$$

with singular values on $\operatorname{diag}(\Sigma)$ sorted in descending order (equivalently, PCA of C). For any $X \in \mathbb{R}^{m \times p}$, the decorrelated representation and its inverse are

$$D = (X - \mu)\,V, \qquad X = D\,V^\top + \mu, \tag{2}$$

where the equality holds when all p components are used. When $r < p$ and the basis is truncated to $V \in \mathbb{R}^{p \times r}$, then $X \approx D\,V^\top + \mu$. For scalability, we use randomized SVD (Halko et al., 2010) calculated on a GPU with target rank $r < p$, substantially reducing runtime and memory. Results for this variant are shown in Figure 4, with additional details and ablations in Appendix C.2.

3.2 Quantization

PCA orders principal components by explained variance. We exploit this ordering to allocate a fixed bit budget across PCA coordinates so that high-variance components receive more bits. The allocation is computed once on a calibration set and reused at inference. We quantize D coordinate-wise to obtain $D^{q_1,\dots,q_d}$, where $q_i \in \mathbb{Z}_{\geq 0}$ is the bit-width assigned to the i-th principal component. Under a global bit budget, we minimize the Frobenius reconstruction error

$$\left\| DV^\top - D^{q_1,\dots,q_k} V^\top \right\|_F^2 . \tag{3}$$

Because right-multiplication by an orthonormal matrix preserves the Frobenius norm, we have

$$\left\| DV^\top - D^{q_1,\dots,q_k} V^\top \right\|_F^2 = \left\| (D - D^{q_1,\dots,q_k}) V^\top \right\|_F^2 = \left\| D - D^{q_1,\dots,q_k} \right\|_F^2 . \tag{4}$$

Thus, optimal bit allocation can be found directly in the decorrelated domain.

We solve the constrained allocation with a simple dynamic programming (DP) algorithm that keeps two tables: (1) the minimum reconstruction error achievable using the first i principal components under a payload of b bits; and (2) a backpointer storing the optimal local decision. Pseudocode and a proof sketch of optimality under these constraints are in Appendix C.10.

Furthermore, inspired by the Microscaling data formats (Rouhani et al., 2023), we quantize groups of *subsequent* PCA coordinates together, each group with shared 16-bit shift and scaling factors. The DP optimizes both the per-group bit-width and group size, restricted to $\{1, 16, 64, 256, 1024\}$ components per group. The total budget equals the sum of payload bits across all coordinates plus per-group shift and scaling factors.

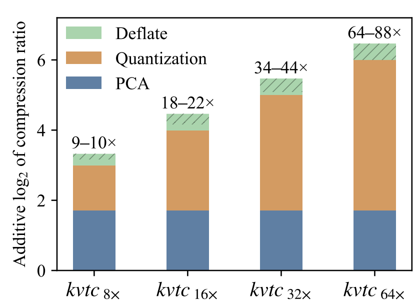


Figure 5: KV cache compression ratios contributed by parts of the `kvtc` pipeline for Llama 3.1 8B. Deflate's variability is marked with black stripes.

An example allocation is shown in Figure 4. As expected, the learned bit-widths decrease monotonically for subsequent principal components. Crucially, the DP assigns zero bits to a substantial number of trailing principal components. This observation motivates reducing the dimensionality during the calculation of PCA, lowering the cost of calibration, and trimming V to the dimensions with nonzero bit-widths for faster compression/decompression during inference.

3.3 Entropy coding

Finally, the quantized values are packed into a single byte array and further compressed using the Deflate algorithm (Wu, 2017). Crucially, we leverage nvCOMP (NVIDIA Corporation, 2020), which enables parallel operation directly on a GPU. This step is lossless, but the added compression ratio is content-dependent.

Table 2: Accuracy of KV cache compression methods. Results within 1 score point of vanilla are in **bold**. See Appendix C.8 for standard error analysis. $\texttt{kvtc}_{y\times}$ denotes `kvtc` set for $y\times$ compression without Deflate.

	Vanilla	$\text{GEAR}_{\text{2-bit}}$	$\text{KIVI}_{\text{2-bit}}$	H_2O	TOVA	xKV	$\texttt{kvtc}_{8\times}$	$\texttt{kvtc}_{16\times}$	$\texttt{kvtc}_{32\times}$	$\texttt{kvtc}_{64\times}$
Llama 3.1 8B										
CR	1	5	5	8	8	1-5	9-10	18-22	34-44	60-88
GSM8K	**56.8**	52.8	52.8	54.3	54.5	**56.6**	**57.0**	**56.9**	**57.8**	**57.2**
MMLU	**60.5**	**59.6**	**59.6**	44.3	44.8	**59.5**	**59.8**	**60.1**	**60.6**	**60.7**
QASPER	**40.4**	**40.4**	39.1	34.3	38.6	35.6	**40.1**	**40.7**	**39.4**	37.8
LITM	**99.4**	96.9	88.8	20.2	1.2	**99.9**	**99.3**	**99.3**	**99.1**	90.2
VT	**99.8**	**99.8**	**98.9**	50.4	**99.7**	**99.8**	**99.1**	**99.1**	**98.9**	95.9
MN-Minitron 8B										
CR	1	5	5	8	8	1-5	10-11	17-21	32-46	53-95
GSM8K	**59.1**	57.9	58.0	55.3	**59.2**	**59.3**	**60.6**	**60.3**	**59.1**	57.8
MMLU	**64.3**	**63.6**	**63.2**	43.5	48.1	63.1	**64.2**	**64.1**	**63.7**	62.1
QASPER	**38.2**	**38.2**	**38.2**	30.0	33.9	34.5	**39.1**	**38.6**	**37.7**	**38.1**
LITM	**99.8**	96.0	86.3	16.6	0.3	**99.6**	**99.4**	**99.3**	86.9	59.5
VT	**99.4**	98.3	96.8	39.2	**99.3**	**99.1**	**98.8**	**98.8**	96.0	93.4
Mistral-NeMo 12B										
CR	1	5	5	8	8	1-5	10-11	17-20	31-43	51-87
GSM8K	**61.9**	59.8	59.7	57.0	60.3	**61.9**	**62.5**	**62.0**	**62.2**	**61.9**
MMLU	**64.5**	**64.0**	**64.3**	45.4	49.0	**63.9**	**64.6**	**64.4**	**63.8**	61.4
QASPER	**38.4**	**38.6**	**38.2**	29.5	36.0	33.5	**37.6**	**37.6**	**37.5**	**38.0**
LITM	**99.5**	96.9	91.9	16.2	8.7	97.9	**99.9**	**99.8**	**99.6**	95.3
VT	**99.8**	**99.4**	98.3	35.2	**99.6**	**99.4**	**99.5**	**99.5**	**98.9**	98.0

4 EXPERIMENTS

Table 3: Reasoning quality (sampling temp 0.6, top-p 95%) of DeepSeek-R1-distilled Qwen2.5. DMS results taken from Łańcucki et al. (2025).

Method	CR	AIME24	AIME25	LCB
		Competition Math		Coding
Qwen 2.5 R1 1.5B				
Vanilla	1	$26.2_{(4.8)}$	$21.7_{(2.9)}$	16.4
$\texttt{kvtc}_{8\times}$	9	$25.4_{(5.7)}$	$24.2_{(4.0)}$	16.1
$\texttt{kvtc}_{16\times}$	18	$27.9_{(6.7)}$	$22.5_{(5.2)}$	13.3
$\text{DMS}_{8\times}$	-	23.3	N/A	16.1
Qwen 2.5 R1 7B				
Vanilla	1	$50.9_{(4.9)}$	$40.8_{(4.3)}$	36.7
$\texttt{kvtc}_{8\times}$	9-11	$52.5_{(3.6)}$	$40.8_{(5.2)}$	36.5
$\texttt{kvtc}_{16\times}$	18-21	$50.9_{(6.8)}$	$38.3_{(5.5)}$	31.6
$\text{DMS}_{8\times}$	-	50.0	N/A	33.4

Models We use models from three families: Llama 3 (Grattafiori et al., 2024), Mistral-NeMo (Mistral AI team, 2024; Sreenivas et al., 2024), and R1-distilled Qwen 2.5 (DeepSeek-AI et al., 2025). The selection includes models ranging from 1.5B to 70B parameters of base instruct, and reasoning kind. Table 1 lists the models along with their KV cache sizes. Notably, MN-Minitron 8B has been pruned from the Mistral-NeMo 12B base, retaining the original KV cache size.

Methods For quantization baselines, we compare our method against KIVI (Zirui Liu et al., 2023) and GEAR (Kang et al., 2024). For eviction baselines, we compare with TOVA (Oren et al., 2024) and H_2O (Zhang et al., 2023). For SVD baselines, we compare our method with xKV (Chang et al., 2025). Finally, we compare to DMS (Łańcucki et al., 2025) on reasoning tasks.

For all methods, we follow the original protocols by performing prefill in vanilla mode and compress the KV cache only after attention has been computed. For every method except xKV, we simulate a sequence of short conversations by running compressing/eviction on the cache every w tokens, where w depends on the method's original sliding window policy. For `kvtc` which is run with $w = 128$ we compress/decompress every 16 tokens, leaving the window in the 112–128 token range. In the case of xKV, we compress only the prefill tokens—providing it with an advantage—since xKV is specifically designed for prefill optimization, and re-computing SVD matrices for newly decoded tokens would be prohibitively time-consuming. For a fair comparison, we only report the prefill compression ratios for non-Qwen models.

Notation-wise, $\texttt{kvtc}^{\blacktriangleright t}_{\mathrm{CR}\times}$ denotes `kvtc` in the default setting, where t is the number of sink tokens excluded from compression, and *CR* is the target compression for the DP. Whenever $t = 4$, we omit this parameter for brevity. Finally, for all methods, we calculate CR only on the compressed tokens, not counting the sliding window tokens.

Table 4: Latency of `kvtc` measured with a simple implementation in HuggingFace Transformers library on an NVIDIA H100 GPU with a Mistral NeMo 12B in bfloat16. TTFT compared in recompute vs decompress and generate first token scenario.

Module	BS=8 CTX=8K Comp	BS=8 CTX=8K Decomp	BS=2 CTX=16K Comp	BS=2 CTX=16K Decomp
Project	153 ms	156 ms	78 ms	75 ms
Quantize	67 ms	37 ms	39 ms	27 ms
Deflate	137 ms	64 ms	66 ms	36 ms
Total	**379 ms**	**267 ms**	**194 ms**	**143 ms**
Vanilla TTFT	**3098 ms**		**1780 ms**	
`kvtc` TTFT	**380 ms**		**208 ms**	

Table 5: Compression of KV cache split into four independent parts, which correspond to four pipeline-parallel instances. We note that the drop in performance for MATH 500 for 20x compression is within $1.5 \times \texttt{stderr}$.

Method	MATH500	NIAH	LITM
Llama 3.3 70B Instruct			
Vanilla	$75.6_{(1.92)}$	100.0	100.0
$\texttt{kvtc}_{8\times}$	$73.2_{(1.98)}$	100.0	100.0
$\texttt{kvtc}_{10\times}$	$74.4_{(1.95)}$	100.0	100.0
$\texttt{kvtc}_{16\times}$	$73.2_{(1.98)}$	100.0	100.0
$\texttt{kvtc}_{20\times}$	$72.6_{(1.99)}$	100.0	100.0

Tasks We evaluate compression effects on Llama 3.1 8B, MN-Minitron 8B and Mistral-NeMo 12B across the following task categories, with results presented in Table 2:

- **Math & Knowledge**: 8-shot Chain of Thought (CoT) GSM8K (Cobbe et al., 2021), 4-shot CoT MMLU (Hendrycks et al., 2021a)
- **Long Context Performance**: 0-shot key-value retrieval task from (Liu et al., 2023a) (denoted LITM), 1-shot RULER (Hsieh et al., 2024) Variable Tracking (denoted VT), and 2-shot Qasper (Question Answering Over Research Papers) (Shaham et al., 2022)

We evaluate R1-distilled models on challenging mathematical competitions AIME 2024-2025 (Art of Problem Solving, 2025) and coding tasks from LiveCodeBench (Jain et al., 2025), with results presented in Table 3. We additionally evaluate `kvtc` with Llama 3.3 70B Instruct on MATH500 (Hendrycks et al., 2021b; Lightman et al., 2023), the key-value retrieval task from (Liu et al., 2023a) and Needle In A Haystack (NIAH) (Kamradt, 2023; Hsieh et al., 2024), with results presented in Table 5. Detailed evaluation protocols can be found in Appendix B. In Appendix C we present ablations and details about parameter choices for `kvtc`.

Calibration Data We sample a 1:1 mixture of short and long documents, with lengths in the 1–8K and 8–32K ranges, respectively. Rotary positional embeddings are removed for calibration; further details are provided in Appendix C.2.

4.1 Results

In all experiments, `kvtc` applies the same compression ratio to both the key and value caches. An ablation of their individual compressibility is presented in Table 7 (Appendix C), suggesting that further adjustments could yield additional gains.

Latency We calculate the latency of elements of the compression pipeline and provide the results in Table 4, in contrast to full re-computation of KV cache. For 8K context length, `kvtc` can reduce time to first token (TTFT) up to $8\times$.

General-Purpose Base Models We evaluate `kvtc` on general-purpose models at the 8–12B scale, featuring three GQA-enabled models (Table 2). The compression ratio of `kvtc` varies due to the data-dependent nature of the Deflate algorithm, which, on average, achieves a compression ratio of approximately $1.23\times$ **on top** of quantization. Crucially, `kvtc` maintains high accuracy across tested tasks, even at substantial compression ratios of $32\times$ and $64\times$. Conversely, quantization methods—GEAR and KIVI—exhibit signs of performance degradation on GSM8K and Lost-in-the-middle tasks only at $5\times$ CR; cache eviction methods such as H_2O and TOVA perform poorly as generic KV cache compressors. We also note that xKV, performs well across most tasks, except for QASPER. Interestingly, in certain cases, `kvtc` at very high compression ratios even surpasses the performance of the vanilla models, likely due to inherent variability in the CoT setup. Crucially, `kvtc` at $16\times$ compression (approximately $20\times$ after Deflate) consistently maintains accuracy within < 1 score point (accuracy or F1, depending on the task) of the vanilla models. The standard errors for these results are reported in Table 11 (Appendix C).

Reasoning Models To assess `kvtc` under more challenging conditions where context plays a critical role, we use complex math and coding tasks (Table 3). Due to high variability, AIME results are averaged over eight independent runs with results reported as $\text{score}_{\texttt{std}}$. On coding tasks, `kvtc` at $10\times$ compression shows minor performance drops of 0.3pp for the 1.5B model and 0.2pp for the 7B model. Notably, the KV cache size of the 1.5B model is already small at 28 MiB per 1K tokens, compared to 128 MiB per 1K tokens for Llama 3.1 8B. A $10\times$ reduction shrinks it further to only 2.8 MiB. We also compare our method against DMS, a state-of-the-art online KV cache compression method. DMS achieves competitive results, and since it employs token eviction, it could potentially be combined with `kvtc` for even lower KV cache footprint.

Multi-GPU Inference To investigate attainable compression ratios for models distributed across multiple GPUs, we evaluate `kvtc` using Llama 3.3 70B (Table 5). The model runs in a pipeline-parallel setting (Hu et al., 2021) on four GPUs, each handling 20 layers. We maintain a local KV cache on each GPU, applying `kvtc` separately following a calibration phase. Performance drops on the MATH500 task are within $1.5 \times \texttt{stderr}$, with accuracy decreasing by 1.2pp at $10\times$ compression and 3.0pp at $20\times$. These experiments did not use Deflate compression, but we anticipate comparable gains to those reported in Table 2. Interestingly, in the distributed 4-GPU setup, KV cache per GPU is smaller for Llama 3.3 70B compared to Llama 3.1 8B.

5 Limitations and Future Work

Online Compression and Combination with Other Methods `kvtc` was targeted for the reduction of time to first token, without affecting the generation time of subsequent tokens. However, the reduction of the KV cache size could be further utilized to optimize the generation part. We leave such research for future work. We note that `kvtc` is compatible with eviction methods such as TOVA, H_2O, and DMS. We also note that `kvtc` could be used to compress the latent state in Multi-head Latent Attention (DeepSeek-AI et al., 2024). However, we do not study such merges here and leave this research for future work.

Field Testing, Larger Models, More Calibration Tokens and Expanded Loss We approximate real use scenarios by testing `kvtc` on a selection of established benchmarks. In our experiments, we limit ourselves to model sizes from 1.5B to 70B parameters. We leave the real user-model deployment with larger models for future work. We have managed to fit between 160K–200K calibration tokens on a single H100 80 GB GPU, leading to a few minutes of PCA calculation with (Halko et al., 2010) algorithm. However, as observed in Figures 4, 7, and 8, increasing the amount of calibration tokens improves the reconstruction error. We leave scaling of `kvtc` beyond 200K tokens for future work. We note that the reconstruction error based on the Frobenius norm is a proxy for measuring how a compression approach will result in the model's downstream performance. We briefly explore the correlation between reconstruction error and downstream performance in Appendix C.5, leaving the exploration of different proxies for future work.

6 Related Work

Quantization-based methods that avoid model fine-tuning offer a straightforward path to KV cache compression (Zhao et al., 2024; Sheng et al., 2023). Works such as KIVI (Zirui Liu et al., 2023) and KVQuant (Hooper et al., 2024) have advanced this direction by developing separate quantization strategies for key and value embeddings. These methods leverage the observation that keys benefit from per-channel quantization, while values are better suited to per-token quantization. Our approach diverges from these methods by first projecting concatenated embeddings from attention layers using SVD matrices derived from a calibration set. Quantization is then applied in this transformed space, with the precision dynamically optimized via dynamic programming bit allocation. While we adopt KIVI's uniform quantization scheme, our application occurs in the SVD-transformed domain. Similar to KVQuant, we apply compression before RoPE (Su et al., 2023) to preserve model quality.

A complementary approach to post-training quantization involves fine-tuning models to adapt to quantized activations. LLM-QAT (Liu et al., 2023b) leverages generations from the pre-quantized model for fine-tuning, while BitDistiller (Du et al., 2024) merges Quantization Aware Training (QAT) (Shao et al., 2024; Bondarenko et al., 2024; Chen et al., 2024; Jacob et al., 2017) with Knowledge Distillation (KD) (Hinton et al., 2015; Gou et al., 2021; Xu et al., 2024a). In contrast, our method eliminates the need for parameter modifications.

SVD has emerged as a straightforward method for removing the redundancy in KV caches and exploiting its low-rank structure. GEAR (Kang et al., 2024) improves quantization through low-rank correction mechanisms, whereas LoRC (Zhang et al., 2024) minimizes computational overhead by directly reducing the rank of key and value matrices. Eigen Attention (Saxena et al., 2024) restructures attention computation by projecting into a truncated subspace defined by SVD, enabling efficient operations. A similar mechanism could be devised for value vectors in `kvtc`, and key vectors for layers that do not employ positional embeddings. GEAR (Kang et al., 2024) improves KIVI quantization through low-rank correction mechanisms. Building on ShadowKV (Sun et al., 2025), SVDq (Yankun et al., 2025) integrates SVD with quantization, leveraging singular value magnitudes for a simple precision allocation; xKV (Chang et al., 2025) aggregates KV caches across multiple layers before decomposition. This work differs in three respects: (i) it models rotational relationships between non-adjacent layers to enable cross-layer concatenation before decomposition; (ii) it selects ranks and bitwidths via a dynamic program under a compression budget; and (iii) it applies entropy coding to the quantized factors. Empirical comparisons and ablations (e.g., treatment of early sink tokens) are reported in Appendix C.

Sparse attention mechanisms provide a complementary paradigm for managing sequence length dimensions by selectively discarding non-essential keys/values during inference. Techniques such as H_2O (Zhang et al., 2023) and TOVA (Oren et al., 2024) employ prioritization strategies to dynamically prune less informative elements from the KV cache. In contrast, chunk-based approaches like Quest (Tang et al., 2024), Landmark Attention (Mohtashami & Jaggi, 2023), and Native Sparse Attention (Yuan et al., 2025) construct compressed representations of the KV cache by partitioning sequences into chunks. Then these methods retrieve only the most critical chunks during attention computation, significantly reducing memory bandwidth requirements by minimizing data transfers from high-bandwidth memory (HBM). Concurrently, dynamic compression techniques such as Dynamic Memory Compression (Nawrot et al., 2024) and Dynamic Memory Sparsification (Łańcucki et al., 2025) optimize KV cache memory usage through pooling/eviction of keys/values. These strategies can be potentially integrated with quantization and SVD methods to achieve higher compression ratios and improved inference latency (Yankun et al., 2025).

Finally, cache management systems address the operational challenges of KV cache handling in production environments. Paged Attention (Kwon et al., 2023) mitigates memory overhead by introducing chunked memory allocation for KV caches. Continuous batching techniques, as implemented in systems like vLLM (Kwon et al., 2023) and FasterTransformer (NVIDIA, 2021), optimized device utilization by enabling parallel processing of multiple sequences. CacheGen (Liu et al., 2024) advanced the field with a distributed framework for long-term KV cache management, incorporating compression, streaming, and cross-node coordination. Our approach extends these systems by integrating fine-grained compression capabilities, enabling token-level compression without compromising distributed architecture advantages.

7 Conclusion

We introduced `kvtc`, a method for compressing KV cache up to 20× with negligible quality degradation, and higher compression ratios of 40× or more possible for specific use cases. We empirically show that key and value caches exhibit substantial redundancy, which `kvtc` exploits through a simple, transform coding pipeline, built around an automatic precision assignment algorithm. We demonstrate effectiveness across both pre-thinking and thinking model families, evaluating models from 1.5B to 70B, and believe that `kvtc` paves the way towards more efficient LLM deployments, lowering the cost of LLM-assisted iterative workflows.

REPRODUCIBILITY

To foster reproducibility of our results, we provide extensive details about the calibration of PCA matrices in Appendix C.2 along with ablations regarding `kvtc` parameters in Appendices C.3 (sink tokens), C.4 (key vs value compressibility), C.5 (amount of calibration data), C.6 (effect of calibration data domain) and C.7 (sliding window). In Appendix B, we present the details about the evaluation setup — tasks, used prompts, and baseline configuration. We note that we utilize LM Eval Harness (Gao et al., 2024) and RULER (Hsieh et al., 2024) for evaluation, which are publicly available. In Appendix C.10 we provide the pseudocode for the dynamic programming precision assignment algorithm along with the sketch of the optimality proof and complexity analysis. We will release the source code after acceptance.

ETHICAL STATEMENT

As a method that aims to improve aspects regarding LLM usage, `kvtc` does not introduce new risks. However, we note that it can amplify existing ones. Therefore, we refer to the existing body of knowledge on the ethical risks of LLM development, such as Ethics Threats in LLM-Based Agents (Gan et al., 2024), potential reversal of safety alignment (Xu et al., 2024b), and more general risks regarding LLMs (Li & Fung, 2025).

REFERENCES

N. Ahmed, T. Natarajan, and K.R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93, 1974. doi: 10.1109/T-C.1974.223784.

Art of Problem Solving. American invitational mathematics examination, 2025. URL https://artofproblemsolving.com/wiki/index.php/American_Invitational_Mathematics_Examination. Art of Problem Solving Wiki.

Yelysei Bondarenko, Riccardo Del Chiaro, and Markus Nagel. Low-rank quantization-aware training for llms, 2024. URL https://arxiv.org/abs/2406.06385.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=M2UzLRoqic.

Chi-Chih Chang, Chien-Yu Lin, Yash Akhauri, Wei-Cheng Lin, Kai-Chiang Wu, Luis Ceze, and Mohamed S. Abdelfattah. xkv: Cross-layer svd for kv-cache compression, 2025. URL https://arxiv.org/abs/2503.18893.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. Efficientqat: Efficient quantization-aware training for large language models, 2024. URL https://arxiv.org/abs/2407.11062.

Yihua Cheng, Kuntai Du, Jiayi Yao, and Junchen Jiang. Do large language models need a content delivery network? *arXiv preprint arXiv:2409.13761*, 2024.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024. URL https://arxiv.org/abs/2403.04132.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers, 2021. URL https://arxiv.org/abs/2105.03011.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Dayou Du, Yijia Zhang, Shijie Cao, Jiaqi Guo, Ting Cao, Xiaowen Chu, and Ningyi Xu. Bitdistiller: Unleashing the potential of sub-4-bit llms via self-distillation, 2024. URL https://arxiv.org/abs/2402.10631.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study, 2025. URL https://arxiv.org/abs/2506.06266.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

Yuyou Gan, Yong Yang, Zhe Ma, Ping He, Rui Zeng, Yiming Wang, Qingming Li, Chunyi Zhou, Songze Li, Ting Wang, Yunjun Gao, Yingcai Wu, and Shouling Ji. Navigating the risks: A survey of security, privacy, and ethics threats in llm-based agents, 2024. URL https://arxiv.org/abs/2411.09523.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, March 2021. ISSN 1573-1405. doi: 10.1007/s11263-021-01453-z. URL http://dx.doi.org/10.1007/s11263-021-01453-z.

J. C. Gower and G. B. Dijksterhuis. *Procrustes Problems*. Oxford University Press, 2004.

V.K. Goyal. Theoretical foundations of transform coding. *IEEE Signal Processing Magazine*, 18 (5):9–21, 2001. doi: 10.1109/79.952802.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, 2010. URL https://arxiv.org/abs/0909.4061.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021a. URL https://arxiv.org/abs/2009.03300.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021b. URL https://arxiv.org/abs/2103.03874.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization, 2024. URL https://arxiv.org/abs/2401.18079.

Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, UK, 2 edition, 2013. ISBN 978-0-521-54823-6.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Yang Hu, Connor Imes, Xuanang Zhao, Souvik Kundu, Peter A. Beerel, Stephen P. Crago, and John Paul N. Walters. Pipeline parallelism for inference on heterogeneous edge computing, 2021. URL https://arxiv.org/abs/2110.14895.

Hugging Face. Math-verify. https://github.com/huggingface/Math-Verify, 2024. A tool for verifying mathematical answers and expressions with advanced parsing capabilities.

Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025a. URL https://github.com/huggingface/open-r1.

Hugging Face. Transformers documentation. https://huggingface.co/docs/transformers/en/index, 2025b. A framework for state-of-the-art machine learning models in text, computer vision, audio, video, and multimodal tasks.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017. URL https://arxiv.org/abs/1712.05877.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/pdf?id=chfJJYC3iL.

Huiqiang Jiang, YUCHENG LI, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=fPBACAbqSN.

Joint Photographic Experts Group. Jpeg 1 standard (iso/iec 10918-1). International Standard 10918, ISO/IEC, 1994. Image compression standard consisting of multiple parts including core coding technology, compliance testing, extensions, and file interchange format.

Greg Kamradt. Llmtest_needleinahaystack: Doing simple retrieval from llm models at various context lengths to measure accuracy. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023. Accessed: 2025-09-24.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm, 2024. URL https://arxiv.org/abs/2403.05527.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. Openassistant conversations – democratizing large language model alignment, 2023. URL https://arxiv.org/abs/2304.07327.

Miles Q. Li and Benjamin C. M. Fung. Security concerns for large language models: A survey, 2025. URL https://arxiv.org/abs/2505.18889.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Prm800k: A process supervision dataset. *arXiv preprint arXiv:2305.20050*, 2023. A dataset containing 800,000 step-level correctness labels for model-generated solutions to MATH problems.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2024. URL https://arxiv.org/abs/2306.00978.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023a. URL https://arxiv.org/abs/2307.03172.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. Cachegen: Kv cache compression and streaming for fast large language model serving, 2024. URL https://arxiv.org/abs/2310.07240.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models, 2023b. URL https://arxiv.org/abs/2305.17888.

Meta. Llama 3.3 evaluation details. https://github.com/meta-llama/llama-models/blob/main/models/llama3_3/eval_details.md, 2024. GitHub repository containing evaluation details for Llama 3.3 models.

Mistral AI team. Mistral nemo: our new best small model, July 2024. URL https://mistral.ai/news/mistral-nemo. Announcement of 12B parameter model with 128k context length, built in collaboration with NVIDIA.

Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers, 2023. URL https://arxiv.org/abs/2305.16300.

Piotr Nawrot, Adrian Lancucki, Marcin Chochowski, David Tarjan, and Edoardo M. Ponti. Dynamic memory compression: Retrofitting llms for accelerated inference, 2024. URL https://arxiv.org/abs/2403.09636.

NVIDIA. Fastertransformer: A fast and efficient transformer implementation. https://github.com/NVIDIA/FasterTransformer, 2021. Apache-2.0 License.

NVIDIA Corporation. nvcomp, 2020. URL https://github.com/NVIDIA/nvcomp. GPU-accelerated compression/decompression library.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, et al. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns, 2024. URL https://arxiv.org/abs/2401.06104.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170, Santa Clara, CA, February 2025. USENIX Association. ISBN 978-1-939133-45-8. URL https://www.usenix.org/conference/fast25/presentation/qin.

Open R1. Openr1-math-220k. https://huggingface.co/datasets/open-r1/OpenR1-Math-220k, 2025. A large-scale dataset containing 220k math problems with verified reasoning traces.

Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, Stosic Dusan, Venmugil Elango, Maximilian Golub, Alexander Heinecke, Phil James-Roxby, Dharmesh Jani, Gaurav Kolhe, Martin Langhammer, Ada Li, Levi Melnick, Maral Mesmakhosroshahi, Andres Rodriguez, Michael Schulte, Rasoul Shafipour, Lei Shao, Michael Siu, Pradeep Dubey, Paulius Micikevicius, Maxim Naumov, Colin Verrilli, Ralph Wittig, Doug Burger, and Eric Chung. Microscaling data formats for deep learning, 2023. URL https://arxiv.org/abs/2310.10537.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for kv cache compression, 2024. URL https://arxiv.org/abs/2408.05646.

Uri Shaham, Elad Segal, Maor Ivgi, Avia Efrat, Ori Yoran, Adi Haviv, Ankit Gupta, Wenhan Xiong, Mor Geva, Jonathan Berant, and Omer Levy. Scrolls: Standardized comparison over long language sequences, 2022. URL https://arxiv.org/abs/2201.03533.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models, 2024. URL https://arxiv.org/abs/2308.13137.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E. Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu, 2023. URL https://arxiv.org/abs/2303.06865.

Sharath Turuvekere Sreenivas, Saurav Muralidharan, Raviraj Joshi, Marcin Chochowski, Ameya Sunil Mahabaleshwarkar, Gerald Shen, Jiaqi Zeng, Zijia Chen, Yoshi Suhara, Shizhe Diao, Chenhan Yu, Wei-Chun Chen, Hayley Ross, Oluwatobi Olabiyi, Ashwath Aithal, Oleksii Kuchaiev, Daniel Korzekwa, Pavlo Molchanov, Mostofa Patwary, Mohammad Shoeybi, Jan Kautz, and Bryan Catanzaro. Llm pruning and distillation in practice: The minitron approach, 2024. URL https://arxiv.org/abs/2408.11796.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference, 2025. URL https://arxiv.org/abs/2410.21465.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024. URL https://arxiv.org/abs/2406.10774.

Maurice Weber, Daniel Y. Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. Redpajama: an open dataset for training large language models. *NeurIPS Datasets and Benchmarks Track*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Yingquan Wu. Deflate compression algorithm, 2 2017. URL https://patents.google.com/patent/US9577665B2/en.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning, 2024. URL https://arxiv.org/abs/2310.06694.

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

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models, 2024a. URL https://arxiv.org/abs/2402.13116.

Zhihao Xu, Ruixuan Huang, Changyu Chen, and Xiting Wang. Uncovering safety risks of large language models through concept activation vector. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 116743–116782. Curran Associates, Inc., 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d3a230d716e65afab578a8eb31a8d25f-Paper-Conference.pdf.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Hong Yankun, Li Xing, Zhen Hui-Ling, Yu Xianzhi, Liu Wulong, and Yuan Mingxuan. Svdq: 1.25-bit and 410x key cache compression for llm attention, 2025. URL https://arxiv.org/abs/2502.15304.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109, 2025. doi: 10.1145/3689031.3696098. URL https://doi.org/10.1145/3689031.3696098.

Jiayi Yuan, Hongyi Liu, Shaochen Zhong, Yu-Neng Chuang, Songchen Li, Guanchu Wang, Duy Le, Hongye Jin, Vipin Chaudhary, Zhaozhuo Xu, Zirui Liu, and Xia Hu. Kv cache compression, but what must we give in return? a comprehensive benchmark of long context capable approaches, 2024. URL https://arxiv.org/abs/2407.01527.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL https://arxiv.org/abs/2502.11089.

Rongzhi Zhang, Kuang Wang, Liyuan Liu, Shuohang Wang, Hao Cheng, Chao Zhang, and Yelong Shen. Lorc: Low-rank compression for llms kv cache with a progressive compression strategy, 2024. URL https://arxiv.org/abs/2410.03111.

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. URL https://arxiv.org/abs/2306.14048.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving, 2024. URL https://arxiv.org/abs/2310.19102.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving, 2024.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi : Plug-and-play 2bit kv cache quantization with streaming asymmetric quantization, 2023. URL https://rgdoi.net/10.13140/RG.2.2.28167.37282.

Adrian Łańcucki, Konrad Staniszewski, Piotr Nawrot, and Edoardo M. Ponti. Inference-time hyper-scaling with kv cache compression, 2025. URL https://arxiv.org/abs/2506.05345.

APPENDIX

A LLM USAGE

LLMs were used to polish the writing of this paper. In particular, they were employed for grammar, punctuation, and additional consistency checking. Moreover, hints from the models were utilized to improve the composition of this work. Models with internet search capability were used to scan for the potential related works. However, they did not replace human evaluation of the search results.

B EVALUATION DETAILS

B.1 TASKS

For evaluation, we utilize Language Model Evaluation Harness (Gao et al., 2024) and RULER (Hsieh et al., 2024) with the Transformers library (Hugging Face, 2025b) serving as an inference backend.

B.1.1 GSM8K

GSM8K (Cobbe et al., 2021) is an established task for the evaluation of the reasoning of non-reasoning models (models without thinking phase (`<think>...</think>`) before answer) (Yang et al., 2025). We evaluate it in an 8-shot CoT setting with few-shot examples from (Wei et al., 2023). Following (Meta, 2024) we allow for the generation of up to 1024 tokens. Task name in LM Eval Harness is `gsm8k_cot`.

GSM8K 8-shot prompt example

```
Q: There are 15 trees in the grove. Grove workers will plant trees
↪  in the grove today. After they are done, there will be 21
↪  trees. How many trees did the grove workers plant today?
A: There are 15 trees originally. Then there were 21 trees after
↪  some more were planted. So there must have been 21 - 15 = 6.
↪  The answer is 6.

Q: If there are 3 cars in the parking lot and 2 more cars arrive,
↪  how many cars are in the parking lot?
A: There are originally 3 cars. 2 more cars arrive. 3 + 2 = 5. The
↪  answer is 5.

Q: Leah had 32 chocolates and her sister had 42. If they ate 35,
↪  how many pieces do they have left in total?
A: Originally, Leah had 32 chocolates. Her sister had 42. So in
↪  total they had 32 + 42 = 74. After eating 35, they had 74 - 35
↪  = 39. The answer is 39.

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason
↪  has 12 lollipops. How many lollipops did Jason give to Denny?
A: Jason started with 20 lollipops. Then he had 12 after giving
↪  some to Denny. So he gave Denny 20 - 12 = 8. The answer is 8.

Q: Shawn has five toys. For Christmas, he got two toys each from
↪  his mom and dad. How many toys does he have now?
A: Shawn started with 5 toys. If he got 2 toys each from his mom
↪  and dad, then that is 4 more toys. 5 + 4 = 9. The answer is 9.

Q: There were nine computers in the server room. Five more
↪  computers were installed each day, from monday to thursday. How
↪  many computers are now in the server room?
A: There were originally 9 computers. For each of 4 days, 5 more
↪  computers were added. So 5 * 4 = 20 computers were added. 9 +
↪  20 is 29. The answer is 29.

Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On
↪  wednesday, he lost 2 more. How many golf balls did he have at
↪  the end of wednesday?
A: Michael started with 58 golf balls. After losing 23 on tuesday,
↪  he had 58 - 23 = 35. After losing 2 more, he had 35 - 2 = 33
↪  golf balls. The answer is 33.

Q: Olivia has $23. She bought five bagels for $3 each. How much
↪  money does she have left?
A: Olivia had 23 dollars. 5 bagels for 3 dollars each will be 5 x 3
↪  = 15 dollars. So she has 23 - 15 dollars left. 23 - 15 is 8.
↪  The answer is 8.

Q: {QUESTION}
A:
```

B.1.2 MMLU

MMLU (Hendrycks et al., 2021a) is a collection of multiple-choice questions spanning 57 subjects. For MMLU evaluation, we use 4-shot `mmlu_flan_cot_fewshot` from LM Eval Harness, allowing for generation of up to 256 tokens. We pick this setup instead of the perplexity-based forward-pass evaluation of MMLU, because our baselines perform a prefill using full precision KV cache. Therefore, for a fair evaluation, we need to make the model generate a non-zero number of tokens before providing the answer, as otherwise the performance would be identical to vanilla.

MMLU 4-shot prompt example

```
The following are multiple choice questions (with answers) about
 ↪  abstract algebra.Q: Statement 1 | Every element of a group
 ↪  generates a cyclic subgroup of the group. Statement 2 | The
 ↪  symmetric group S_10 has 10 elements.
(A) True, True (B) False, False (C) True, False (D) False, True
A: Let's think step by step. A cyclic group is a group that is
 ↪  generated by a single element. Hence a subgroup generated by a
 ↪  single element of a group is cyclic and Statement 1 is True.
 ↪  The answer is (C).

Q: The symmetric group $S_n$ has $
actorial{n}$ elements, hence it is not true that $S_{10}$ has 10
 ↪  elements.
Find the characteristic of the ring 2Z.
(A) 0 (B) 3 (C) 12 (D) 30
A: Let's think step by step. A characteristic of a ring is R is $n$
 ↪  if the statement $ka = 0$ for all $a\in 2Z$ implies that $k$ is
 ↪  a multiple of $n$. Assume that $ka = 0$ for all $a\in 2Z$ for
 ↪  some $k$. In particular $2k = 0$. Hence $k=0$ and $n=0$. The
 ↪  answer is (A).

Q: Statement 1| Every function from a finite set onto itself must
 ↪  be one to one. Statement 2 | Every subgroup of an abelian group
 ↪  is abelian.
(A) True, True (B) False, False (C) True, False (D) False, True
A: Let's think step by step. Statement 1 is true. Let $S$ be a
 ↪  finite set. If $f:S
ightarrow S$ is a onto function, then $|S| = |f(S)|$. If $f$ was
 ↪  not one to one, then for finite domain $S$ the image would have
 ↪  less than $S$ elements, a contradiction.
Statement 2 is true. Let $G$ be an abelian group and $H$ be a
 ↪  subgroup of $G$. We need to show that $H$ is abelian. Let $a,b
 ↪  \in H$. Then $a,b \in G$ and $ab=ba$. Since $G$ is abelian,
 ↪  $ab=ba$. Since $H$ is a subgroup of $G$, $ab \in H$. Therefore,
 ↪  $ab=ba$ and $H$ is abelian. The answer is (A).

Q: Statement 1 | If aH is an element of a factor group, then |aH|
 ↪  divides |a|. Statement 2 | If H and K are subgroups of G then
 ↪  HK is a subgroup of G.
(A) True, True (B) False, False (C) True, False (D) False, True
A: Let's think step by step. Statement 2 is false. Let $H$ be a
 ↪  subgroup of $S_3$ generated by the cycle $(1,2)$ and $K$ be a
 ↪  subgroup of $S_3$ generated by the cycle $(1,3)$. Both $H$ and
 ↪  $K$ have two elements, the generators and the identity. However
 ↪  $HK$ contains cycles (1,2), (1,3) and (2,3,1), but the inverse
 ↪  of (2,3,1) is (2,1,3) and it does not belong to HK, hence HK is
 ↪  not a subgroup. The answer is (B).

Q: {QUESTION}
A: Let's think step by step.
```

B.1.3 LOST IN THE MIDDLE

Lost in the Middle (Liu et al., 2023a) is evaluated in a 0-shot 100-keys (300 keys for Llama 3.3 70B) setup, allowing generation of 64 tokens using the prompt presented below. We implement the evaluation using the LM Eval Harness framework and utilize the UUID strings and code from (Liu et al., 2023a). This benchmark allows for methodical testing of the model's ability to access the input context. As an evaluation metric, we utilize the exact match between the UUID returned by the model and the gold answer.

Lost in the middle question example (base models)

```
Extract the value corresponding to the specified key in the JSON object below.

JSON data:
{"1afcec1f-1acd-42e3-b833-e7882d5daada": "25f1a78d-a2f6-4c7d-8bd6-51226b263cbe",
 "94071d67-86df-455c-8ee9-691e492ff740": "0d7ba717-e034-410e-88ab-c13d37cc6499",
 "88b322bb-571c-4e55-9934-aa8df11b3349": "c54095cf-9931-460b-8a6b-e1f09afb2f72",
...
 "5a729e1f-6956-4c1d-b024-10b317ed5657": "cea37ae5-84e1-4deb-b4c6-19d04134d664",
 "aaed65fc-f80c-4090-a0a9-90592140b9de": "ffc2e314-2d0f-4b20-be32-916ba96d1ea9",
 "90b7fe08-8708-451a-badf-34cabe7930a4": "7bebff53-05c7-4ca3-9314-bca68bd65c04"}
 "1afcec1f-1acd-42e3-b833-e7882d5daada":
```

Lost in the middle question example (Llama 3.3 70B instruct)

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023
Today Date: 26 Jul 2024

<|eot_id|><|start_header_id|>user<|end_header_id|>

Extract the value corresponding to the specified key in the JSON object below.

JSON data:
{"2a85047d-fe61-4c53-8844-1d85668d6a7d": "a4852ee7-d94f-40ce-8c2f-f14e95377e79",
 "1698320e-2ba6-499f-b119-b8ffd74d53db": "e212083b-22f5-4d27-87d3-5c3cfcf9542f",
 ...
 "90ef90b0-7972-46d0-9a73-4b07be2f5aae": "ebb84b27-9156-4d23-8a7d-a34aef606f28",
 "c1736979-584d-4b93-8e25-a206770fcdae": "dcb5aace-7fb2-4288-bfc2-c5faacf89469"}
  What is the value associated with the key "2a85047d-fe61-4c53-8844-1d85668d6a7d"? Answer using the
  ↪  following format:

`The value associated with the key "2a85047d-fe61-4c53-8844-1d85668d6a7d" is ANSWER_HERE.`

Where ANSWER_HERE is the value associated with the key
↪  "2a85047d-fe61-4c53-8844-1d85668d6a7d".<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

B.1.4 Variable Tracking

Variable Tracking is evaluated in 1-shot (with a relatively short example shown below) using RULER (Hsieh et al., 2024) with context length 8K, limiting the generation to 128 tokens. The benchmark tests the model's ability to track variable assignments across unrelated contexts.

Variable Tracking prompt and question example

```
Memorize and track the chain(s) of variable assignment hidden in the following text.

The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 VAR JUP = 97498 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back
 ↪  again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 VAR AGD = VAR JUP  The grass is green. The sky is blue. The sun is yellow. Here we go. There and
 ↪  back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 VAR KCB = VAR AGD  The grass is green. The sky is blue. The sun is yellow. Here we go. There and
 ↪  back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 VAR LJP = VAR KCB  The grass is green. The sky is blue. The sun is yellow. Here we go. There and
 ↪  back again.
 The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again.
 VAR LFP = VAR LJP  The grass is green. The sky is blue. The sun is yellow. Here we go. There and
 ↪  back again.

Question: Find all variables that are assigned the value 97498 in the text above. Answer: According
↪  to the chain(s) of variable assignment in the text above, 5 variables are assgined the value
↪  97498, they are:  JUP AGD KCB LJP LFP



Memorize and track the chain(s) of variable assignment hidden in the following text.

...
Question: Find all variables that are assigned the value 79092 in the text above. Answer: According
↪  to the chain(s) of variable assignment in the text above, 5 variables are assgined the value
↪  79092, they are:
```

B.1.5 NEEDLE IN A HAYSTACK

Needle In A Haystack (NIAH) (Kamradt, 2023) is evaluated 0-shot using RULER (Hsieh et al., 2024) with context length 100K, limiting the generation to 128 tokens. The benchmark tests the model's ability to retrieve information hidden in long text. The name of the task in the RULER is `niah_single_2`.

NIAH prompt example

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023\nToday Date: 26 Jul 2024

<|eot_id|><|start_header_id|>user<|end_header_id|>

A special magic number is hidden within the following text. Make sure to memorize it. I will quiz you
↪  about the number afterwards.

<essay text prefix>
<needle>
<essay text suffix>
What is the special magic number for abrasive-pathology mentioned in th
e provided text? The special magic number for abrasive-pathology mentioned in the provided text
↪  is<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

B.1.6 QASPER

Qasper (Dasigi et al., 2021; Shaham et al., 2022) is evaluated in 2-shot using LM Eval Harness task `scrolls_qasper` with full autoregressive generation and F1 score (same reasons as with MMLU evaluation). We limit the generation to 128 tokens. This benchmark evaluates the model's ability to answer questions about a paper presented as part of the input.

B.1.7 AIME 2024-2025

We implement AIME evaluation using LM Eval Harness. Following (Łańcucki et al., 2025) we utilize prompts adopted from the Open-R1 repository (Hugging Face, 2025a) and limit the generation to 30K tokens. AIME competitions are popular for the evaluation of reasoning models such as DeepSeek R1 (DeepSeek-AI et al., 2025). To check the correctness of an answer, we utilize the following code with Math-Verify (Hugging Face, 2024):

AIME 2024-2025 evaluation code

```
from math_verify.metric import math_metric
from math_verify.parser import LatexExtractionConfig,
↪  ExprExtractionConfig

def grade_answer(problem, model_answer):
    gold_is_latex = False
    verify_func = math_metric(
        gold_extraction_target=(
            LatexExtractionConfig() if gold_is_latex else
             ↪  ExprExtractionConfig(),
        ),
        pred_extraction_target=(ExprExtractionConfig(),
         ↪  LatexExtractionConfig()),
        aggregation_function=max,
        precision=6,
    )
    gold_answer = problem["answer"]

    try:
        with timeout(seconds=30): # custom class to throw and
         ↪  exception if code does not complete under 30 seconds
            grade, extracted_answers = verify_func([gold_answer],
             ↪  [model_answer])
            return grade == 1
    except:
        return False
```

AIME 2024-2025 prompt

```
<|begin_of_sentence|><|User|>Solve the following math problem
↪  efficiently and clearly:

- For simple problems (2 steps or fewer):
Provide a concise solution with minimal explanation.

- For complex problems (3 steps or more):
Use this step-by-step format:

## Step 1: [Concise description]
[Brief explanation and calculations]

## Step 2: [Concise description]
[Brief explanation and calculations]

...

Regardless of the approach, always conclude with:

Therefore, the final answer is: $\boxed{answer}$. I hope it is
↪  correct.

Where [answer] is just the final number or expression that solves
↪  the problem.

Problem: {PROBLEM}

<|Assistant|><think>
```

B.1.8 LiveCodeBench

For LiveCodeBench (Jain et al., 2025) evaluation, we utilize the official repository and, following (Łańcucki et al., 2025), limit the generation to 16K tokens and data range from 2024-08-01 to 2025-01-31. The benchmark consists of problems from sites like leetcode.com and codeforces.com.

B.1.9 MATH500

For evaluation on MATH500 (Lightman et al., 2023) (subset of MATH (Hendrycks et al., 2021b) introduced by (Lightman et al., 2023)), we limit the generation to 5120 tokens following (Meta, 2024). This is the only task where we change the 128 window of `kvtc` recent tokens to 256 one. We utilize the following prompt adopted from MATH500 evaluation, and the following code optimized for reproduction of Llama 3.3 70B results without the LLM judge:

MATH500 eval code

```python
from math_verify import parse, verify

def answer_normalize(answer: str) -> str:
    answer = answer.split(r"\boxed{")[-1].split("}$")[0].strip()
    answer = answer.replace(r"\left", "")
    answer = answer.replace(r"\right", "")
    answer = answer.replace(r"\begin{align}", "")
    answer = answer.replace(r"\end{align}", "")
    answer = answer.replace(r"\begin{equation}", "")
    answer = answer.replace(r"\end{equation}", "")
    answer = answer.replace(" ", "")
    answer = answer.replace(r"\$", "")
    if answer.startswith(r"\text"):
        answer = answer.replace(r"\text{", "")
        answer = answer.replace(r"}", "")

    if answer.startswith(r"x\in"):
        answer = answer.replace(r"x\in", "")

    if answer.startswith(r"y="):
        answer = answer.replace(r"y=", "")

    return answer


def compare_answers(answer: str, model_answer: str) -> bool:
    gold = parse(answer)
    model = parse(model_answer)
    res = verify(gold, model)
    if not res:
        answer = answer_normalize(answer)
        model_answer = answer_normalize(model_answer)
        print(answer, model_answer)
        gold = parse(answer)
        model = parse(model_answer)
        if not verify(gold, model): # sometimes fails for improper expressions
            res = verify(answer, model_answer)
            gold = answer
            model = model_answer
        if res:
            return True
        else:
            return False
    else:
        return res
```

MATH500 prompt

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023
Today Date: 26 Jul 2024

<|eot_id|><|start_header_id|>user<|end_header_id|>

Solve the following math problem efficiently and clearly:

- For simple problems (2 steps or fewer):
Provide a concise solution with minimal explanation.

- For complex problems (3 steps or more):
Use this step-by-step format:

## Step 1: [Concise description]
[Brief explanation and calculations]

## Step 2: [Concise description]
[Brief explanation and calculations]

...

Regardless of the approach, always conclude with:

Therefore, the final answer is: $\boxed{answer}$. I hope it is correct.

Where [answer] is just the final number or expression that solves the problem.

Problem: {PROBLEM}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

B.2 BASELINES CONFIGURATION

B.2.1 KIVI

We follow (Zirui Liu et al., 2023, Section 4.1) and use `group size` = 32 and `residual length` = 128, with 2-bits per key and 2-bits per value. We utilize the official implementation.

B.2.2 GEAR

We follow (Kang et al., 2024) github repository and set underlying quantization to KIVI with `group size` = 64, `streaming gap` = 64, 2-bit keys and values. Additionally, we set the rank of key/value correction to 4 for prefill and 2 for generation. We utilize the official implementation.

B.2.3 xKV

We follow (Chang et al., 2025). For Llama 3.1 8B we group layers by 4 and set the pre-RoPE key rank to 512 (compression ration 8) and value rank to 768 (compression rank $5\frac{1}{3}$). For Mistral NeMo and MN-Minitron, we group layers by 5 (those models have 1.25 more layers than Llama 3.1 8B) and set the pre-RoPE key rank to 640 (compression ratio 8) and value rank to 960 (compression rank $5\frac{1}{3}$). We utilize the official implementation.

B.2.4 H2O

We utilize the official implementation of (Zhang et al., 2023) and set the recent and heavy hitter fractions to $\frac{1}{16}$ of (input + max possible output size). This results in up to an 8x compression ratio. Lower compression ratios occur when the model produces shorter outputs than the maximal specified value. We note that this can give an advantage to H2O over other methods.

B.2.5 TOVA

We utilize the official implementation of (Oren et al., 2024) and set the max cache size to $\frac{1}{8}$ of (input + possible output size). This results in up to an 8x compression ratio. Lower compression ratios occur when the model produces shorter outputs than the maximal specified value. We note that this can give an advantage to TOVA over other methods.

B.2.6 DMS

We copy the results from the Inference-Time Hyper-Scaling (Łańcucki et al., 2025).

B.3 HARDWARE

Experiments were run on a node with 8 $\times$ NVIDIA H100 GPUs (80 GB each). All jobs finished within 4 h, except MMLU and Llama-3.3-70B ($\leq$ 8 h) and xKV ($\leq$ 12 h). For baselines (KIVI, GEAR, xKV, TOVA, H2O) we used the authors' public code. All runs used batch size 1 (except Qwen evaluations) to prevent padding that could bias some of the baselines.

C Ablations and additional details

We present additional details and ablations regarding `kvtc` in the following appendices:

- Appendix C.1 — elaboration on our choice of PCA for dimensionality reduction and decorrelation by providing mathematical and empirical justification.
- Appendix C.2 — additional details about the hyperparameters of `kvtc`, along with justifications and references to relevant ablation studies.
- Appendix C.3 — omitting sink tokens for KV cache compression can result in significant gains at higher compression ratios.
- Appendix C.4 — the difference in compressibility between keys and values suggests that long-context retrieval tasks may benefit from using higher precision for keys.
- Appendix C.5 — increasing the amount of calibration data helps preserve performance at higher compression ratios, while smaller amounts remain competitive for lower compression.
- Appendix C.6 — influence of the calibration data domain on downstream performance.
- Appendix C.8 — results from Table 2, with standard errors computed by LM Eval Harness (Gao et al., 2024).
- Appendix C.9 — sizes of PCA projection matrices (stored per model), with an emphasis on the fact that their size is only a small fraction of the model parameters.
- Appendix C.10 — pseudocode for the Dynamic Programming (DP) precision assignment algorithm, along with complexity analysis and a sketch of the optimality proof.

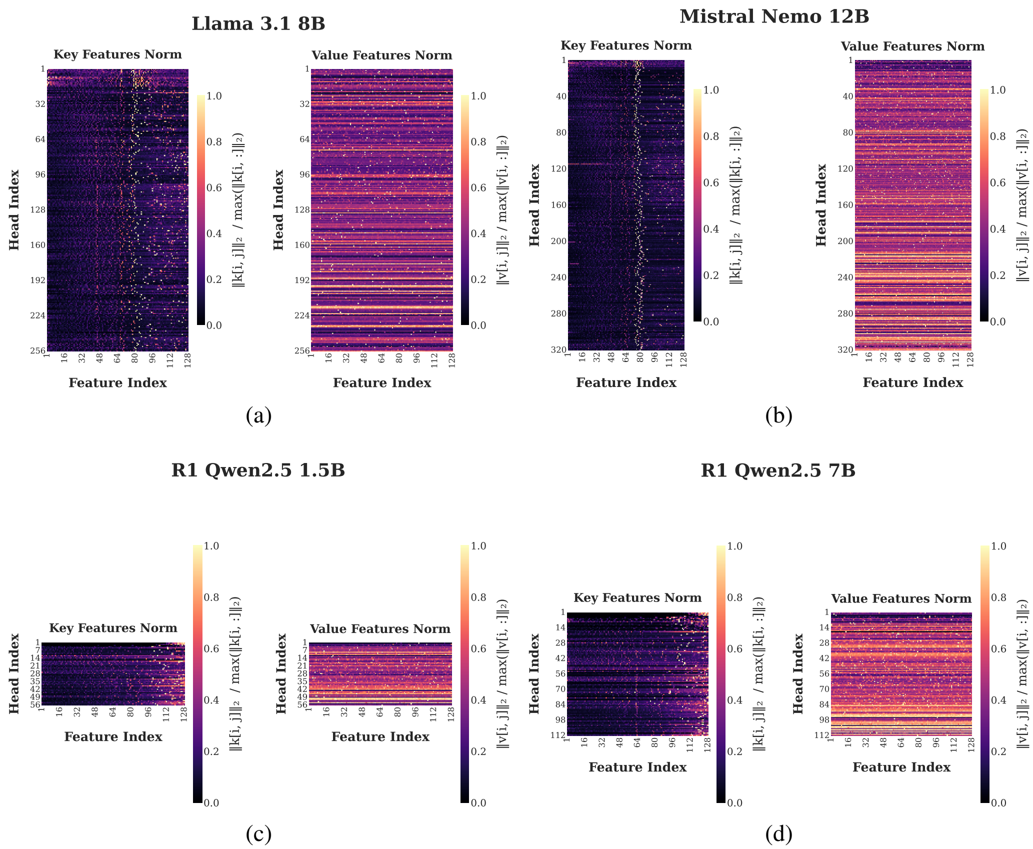


Figure 6: Norms of pre-RoPE key and value channels averaged across 10 calibration documents, each between 1K and 8K tokens. For each key/value head and each channel we present the average norm divided by the highest average channel norm within the same head, that is $\mathrm{heatmap}_{i,j} = \mathrm{norm}_{i,j} / \max_b \{\mathrm{norm}_{i,b}\}$. We hypothesize that the repeating pattern observed for key heads is an effect of Rotary Positional Embedding (Su et al., 2023), which uses different frequencies for different pairs of consecutive channels. This can incentivize the model to focus more on particular channels, while omitting other ones.

C.1 PCA MOTIVATION

Figure 2 shows that a simple orthogonal transformation can make keys/values produced by different attention heads much more similar. What is more, Figure 6 shows high potential for dimensionality reduction within most of the keys and some of the values. As noted in the main body of the paper, this motivates our choice of PCA as a dimensionality reduction/decorrelation method. In particular, note that for

$$S = \{[v^T,\ v^T R]^T : v \in A\},$$

where $R \in \mathbb{R}^{d \times d}$ is orthogonal and A is a finite subset of $\mathbb{R}^d$. If $\{u_i\}_i$ explains all the variance of A, then $\{[u_i^T,\ u_i^T R]^T\}_i$ explains all variance of S.

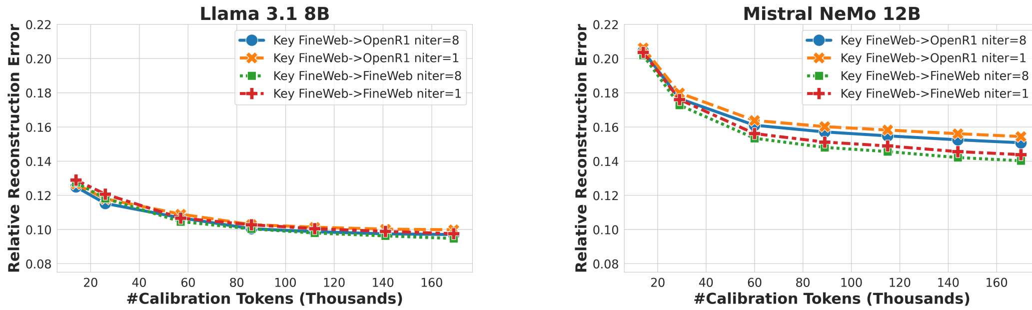
Another motivation comes from the work on efficient attention kernels (Jiang et al., 2024). To be more precise, from the observation that different attention heads can show similar attention patterns. We note that in a simplified case, where keys are equal to queries and we assume the exact equality of dot products that create the attention patterns, this results in the key spaces being equal up to an orthogonal transformation. The proof follows from the uniqueness of vector realizations for a given Gram matrix (matrix of inner products) and can be found in (Horn & Johnson, 2013).

C.2 PCA CALCULATION PARAMETERS

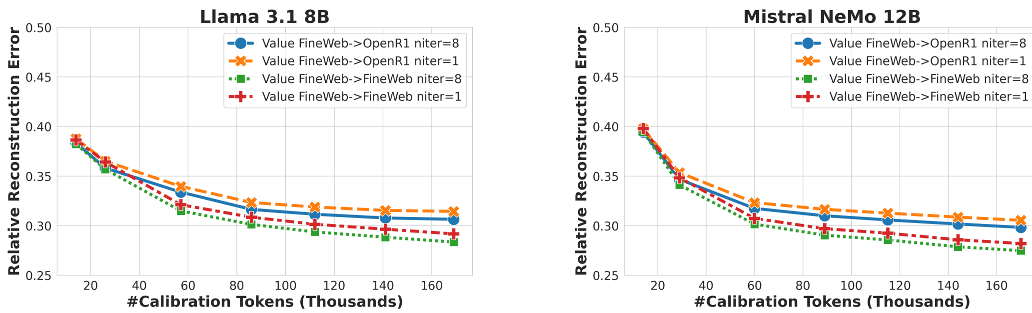
As mentioned in the main text, we utilize 8 iterations of the randomized algorithm from (Halko et al., 2010) for PCA. We utilize 160K calibration tokens for Llama 3.1 8B, Llama 3.3 70B instruct, Mistral NeMo 12B, and MN-Minitron 8B with a dimensionality cut-off of 10K. This choice is motivated by memory efficiency and the results presented in Figures 4, 7 and 8, where the initial boost from the increase in the number of calibration tokens from 10K to 100K is relatively large compared to the boost attained when increasing from 100K to 160K. We leave the exploration of larger calibration sets for future work. In Appendix C.5 we ablate the influence of the amount of the calibration data on downstream results. In Appendix C.9 we provide the sizes of the projection matrices, noting that they are relatively small when compared to the model size.

For Qwen models, due to their smaller number of KV heads, we utilize 200K calibration tokens and dimensionality reduction of 8K. For all models, we utilize a 50/50 mixture of FineWeb (Penedo et al., 2024) and OpenR1Math traces (R1, 2025) due to the duality of our benchmarks (reasoning and general purpose; for ablation on data source vs performance (see Appendix C.6 for an ablation on calibration data source). We take samples from both datasets with the only filters being minimum and maximum length (1K and 32K, respectively) for both datasets and quality score (≥ 0.95) for FineWeb (the quality score is attached to the dataset). Additionally, we ensure that the number of tokens from documents below 8000 and above 8000 tokens is roughly the same (except in the MN-Minitron case, as this model supports up to 8K context length). Token counts and cutoffs are chosen so that a single calculation of PCA fits on a single H100 GPU with 80GB of memory and completes within 10 minutes (for details see Appendix C.5).

We emphasize that for a given model, we use the same PCA matrix for all compression ratios. The only change between compression ratios is the precision assignment, which is done automatically via a dynamic programming algorithm. Additionally, we note that the manual adjustment needed by a practitioner to adapt `kvtc` to a new model is limited to choosing the initial PCA dimensionality cutoff (if the practitioner decides to use the efficient algorithm by (Halko et al., 2010)), the number of samples that the chosen PCA implementation can handle (based on GPU/CPU memory) and the calibration sample choice if special scenarios are desired for increased compression. In this paper, we note that across a wide range of tasks, the choice of a 50/50 mixture of both short and long context FineWeb (Penedo et al., 2024) (for generic text) and OpenR1Math (R1, 2025) (for thinking traces) data is sufficient for Llama, Mistral, and R1Qwen models for a variety of applications. In particular, we note that our method is not harder to adjust than xKV or SVDq, due to automatic precision assignment via dynamic programming. In particular, instead of aiming for the best reconstruction error for a specific compression ratio, the user can easily alter the algorithm to aim for the highest compression ratio within a given reconstruction error constraint.

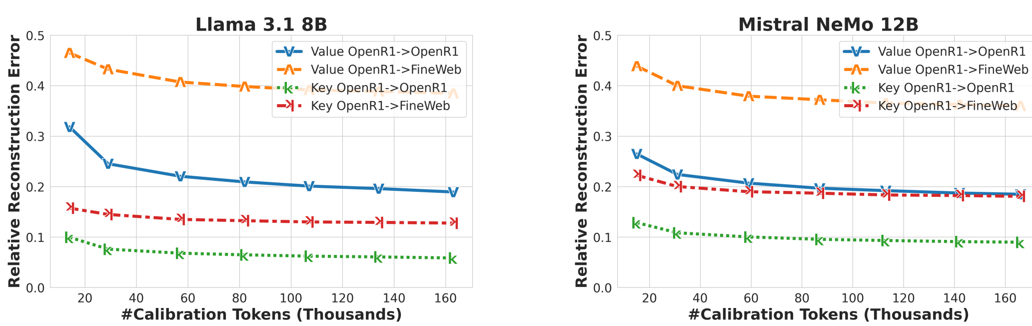


(a) (b)



(c) (d)

Figure 7: Relative reconstruction error when calibrating `kvtc` decorrelation step - ablation of the number of algorithm (Halko et al., 2010) iterations. Figures (a)-(b) show key reconstruction error, whereas (c)-(d) show value reconstruction error. Other parameters as in Figure 4. We note that the larger number of iterations provides slight improvements.



(a) (b)

Figure 8: Relative reconstruction error when calibrating `kvtc` decorrelation step on OpenR1Math (R1, 2025) traces with the error calculation on FineWeb (Penedo et al., 2024). Parameters as in Figure 4. We note that OpenR1Math traces were published after the release of Llama 3.1 8B and Mistral NeMo 12B, and possibly due to their specificity result in higher generalization error.

C.3 EXCLUDING SINK TOKENS FROM COMPRESSION

We consider the first four tokens of key and value caches (i.e., tokens at positions 0, 1, 2, and 3) to be sink tokens, motivated by the experimental results reported by (Xia et al., 2024). Reducing the dimensionality of key and value caches with PCA causes larger information loss of the sink tokens than the remaining ones, as shown in Figure 4. In order to assess the influence, we compare two setups: one that excludes four sink tokens from compression (denoted $\texttt{kvtc}^{\blacktriangleright 4}$) and one that compresses them along with other tokens (denoted $\texttt{kvtc}^{\blacktriangleright 0}$). The results are shown in Table 6. High compression ratio of $64\times$ drastically reduces downstream task scores for the Llama 3.1 8B in the $\texttt{kvtc}^{\blacktriangleright 0}$ case; for MN-Minitron 8B it causes regression on the long context tasks (Lost-in-the-middle and Variable Tracking) when compared with $\texttt{kvtc}^{\blacktriangleright 4}$.

Table 6: Ablation on the skipping compression of the first four tokens. We note that the difference starts to be visible with larger compression ratios, and that it is in favor of skipping compression of potential attention sinks (Xiao et al., 2024). Results are presented as $\mathrm{score}_{\mathrm{stderr}}$ where stderr is bootstraped by LM Evaluation Harness (where available) (Gao et al., 2024).

Model	Method	CR	GSM8K	MMLU	QASPER	LITM 100	RULER-VT
			Math & Knowledge		Long Context		
Llama 3.1 8B	$\texttt{kvtc}^{\blacktriangleright 0}_{16\times}$	19-22	$56.8_{1.4}$	$60.3_{0.4}$	40.2	$99.2_{0.1}$	$98.2_{0.4}$
	$\texttt{kvtc}^{\blacktriangleright 4}_{16\times}$	18-22	$56.9_{1.4}$	$60.1_{0.4}$	40.7	$99.3_{0.1}$	$99.1_{0.3}$
	$\texttt{kvtc}^{\blacktriangleright 0}_{64\times}$	76-90	$1.6_{0.3}$	$9.9_{0.2}$	27.6	$0.0_{0.0}$	$61.8_{1.5}$
	$\texttt{kvtc}^{\blacktriangleright 4}_{64\times}$	60-88	$57.2_{1.4}$	$60.7_{0.4}$	37.8	$90.2_{0.4}$	$95.9_{0.6}$
MN-Minitron 8B	$\texttt{kvtc}^{\blacktriangleright 0}_{64\times}$	79-97	$59.5_{1.4}$	$61.9_{0.4}$	37.9	$55.1_{0.6}$	$91.5_{0.9}$
	$\texttt{kvtc}^{\blacktriangleright 4}_{64\times}$	53-95	$57.8_{1.4}$	$62.1_{0.4}$	38.1	$59.5_{0.6}$	$93.4_{0.8}$

C.4 Separate adjustment of compression ratios for key and value caches

All experimental results of `kvtc` (unless stated otherwise) have been obtained with 1:1 compression of keys and values. We present additional results of manually adjusting the compression ratio separately for keys and values (Table 7). The results suggest that for long-context retrieval tasks, the value cache could be compressed more than the key cache. We attribute this phenomenon to the necessity to precisely attend to selected tokens in the cache, which hinges on the high accuracy of key vectors. On the other hand, stronger compression of values shows a noticeable degradation on the GSM8K and MMLU tasks.

Table 7: Ablation of key/value compressibility with `kvtc`. We use the same `kvtc` configuration as in Table 2, but independently change key and value lossy compression rates. To denote that value compression ratio was set to 32 and key was set to 64 we use $\texttt{kvtc}^{\blacktriangleright 0}_{\frac{\mathbf{k:64\times}}{\mathrm{v:}32\times}}$

Method	CR	GSM	MMLU	QASP	LITM	VT
Llama 3.1 8B						
$\texttt{kvtc}^{\blacktriangleright 4}_{\frac{\mathbf{k:256\times}}{\mathrm{v:}32\times}}$	55-80	$57.1_{1.4}$	57.4	36.6	$48.6_{1.6}$	$91.9_{0.9}$
$\texttt{kvtc}^{\blacktriangleright 4}_{\frac{\mathrm{k:}32\times}{\mathbf{v:256\times}}}$	55-77	$56.8_{1.4}$	57.5	36.3	$71.9_{1.4}$	$95.9_{0.6}$
Mistral-NeMo 12B						
$\texttt{kvtc}^{\blacktriangleright 0}_{\frac{\mathbf{k:256\times}}{\mathrm{v:}32\times}}$	69-79	$62.2_{1.3}$	61.7	34.6	$73.2_{1.4}$	$90.4_{0.9}$
$\texttt{kvtc}^{\blacktriangleright 0}_{\frac{\mathrm{k:}32\times}{\mathbf{v:256\times}}}$	69-77	$57.1_{1.4}$	59.8	35.2	$77.5_{1.3}$	$89.5_{1.0}$

C.5 Calibration data tokens vs downstream performance

We test how the amount of calibration data affects calibration times and the downstream performance. From Figures 7 and 8 we already know that increasing the amount of the calibration data can bring down the reconstruction error. The question that remains is how such an increase correlates with the downstream performance. In Table 8 we show that an increase in calibration data clearly benefits a high compression ratio of $256\times$, with more moderate and quickly diminishing returns for smaller ($64\times$, $32\times$) compression ratios. This is a positive result, as it allows for trading calibration stage complexity for improved downstream performance, with 40K token budget bringing already competitive results for $64\times$ ratio. We additionally note that PCA calibration can be completed within 1.5 minutes for 160K tokens using (Halko et al., 2010) algorithm, and that the DP calculation time (performed once per model and compression ratio) can be finalized within 8 minutes.

Table 8: Ablation of the number of tokens used for `kvtc` calibration, along with respective PCA and DP calibration times. PCA calibration was performed using single H100 80GB GPU, calculation of DP tables (except simulation of quantization) was offloaded to the node cpu. We additionally limit DP calibration data to first 32K tokens, therefore we do not see increase in dp time after 40k calibration tokens. All calibration datapieces come from a 50/50 mixture of FineWeb (Penedo et al., 2024) and OpenMathR1 (R1, 2025) traces between 1K and 8K tokens.

Method	Data	PCA Calib	DP Calib	CR	GSM	MMLU	QASP	LITM	VT
				Mistral-NeMo 12B					
$\texttt{kvtc}_{32\times}$	20K	41.3s	6.5 min	31-42	$63.0_{1.3}$	$63.9_{0.4}$	34.8	$97.3_{0.2}$	$98.9_{0.3}$
$\texttt{kvtc}_{64\times}$			5.4 min	63-87	$63.9_{1.3}$	$61.7_{0.4}$	31.8	$88.8_{0.4}$	$96.9_{0.5}$
$\texttt{kvtc}_{256\times}$			3.9 min	148-340	$59.6_{1.4}$	$51.5_{0.4}$	23.3	$10.4_{0.4}$	$66.8_{1.5}$
$\texttt{kvtc}_{32\times}$	40K	47.8s	7.2 min	31-43	$64.2_{1.3}$	$63.0_{0.4}$	35.3	$98.7_{0.1}$	$99.5_{0.2}$
$\texttt{kvtc}_{64\times}$			6.1 min	63-88	$63.7_{1.3}$	$61.9_{0.4}$	32.4	$95.7_{0.3}$	$97.8_{0.5}$
$\texttt{kvtc}_{256\times}$			4.6 min	148-344	$58.3_{1.4}$	$50.1_{0.4}$	23.7	$9.2_{0.4}$	$70.6_{1.4}$
$\texttt{kvtc}_{32\times}$	80K	60.4s	7.2 min	31-43	$63.6_{1.3}$	$63.9_{0.4}$	36.0	$98.0_{0.2}$	$99.3_{0.3}$
$\texttt{kvtc}_{64\times}$			6.1 min	63-88	$63.3_{1.3}$	$62.0_{0.4}$	32.2	$94.5_{0.3}$	$97.8_{0.5}$
$\texttt{kvtc}_{256\times}$			4.6 min	148-339	$60.5_{1.3}$	$51.7_{0.4}$	23.3	$13.6_{0.4}$	$83.7_{1.2}$
$\texttt{kvtc}_{32\times}$	160K	85.6s	7.2 min	31-43	$63.2_{1.3}$	$64.3_{0.4}$	36.6	$99.5_{0.1}$	$99.4_{0.3}$
$\texttt{kvtc}_{64\times}$			6.1 min	63.2-87	$64.6_{1.3}$	$62.2_{0.4}$	32.8	$96.9_{0.2}$	$97.7_{0.5}$
$\texttt{kvtc}_{256\times}$			4.6 min	148-341	$61.4_{1.3}$	$55.5_{0.4}$	24.3	$34.2_{0.6}$	$79.5_{1.3}$

C.6 Calibration data domain vs downstream performance

To examine how the calibration data domain influences downstream performance, we prepare two additional versions of `kvtc` for Mistral Nemo 12B: one using only FineWeb data and the other using only OpenR1Math data. Table 9 shows that using OpenR1Math calibration data better maintains MMLU and key-value retrieval scores than FineWeb at higher compression rates. This improvement is likely related to the question-think-answer structure of OpenR1Math, which may be more aligned with the evaluation tasks, compared to the more general web collection nature of FineWeb. However, for $32\times$ compression, both choices remain competitive. We note that for $256\times$ compression, the 50/50 mixture of FineWeb and OpenR1Math results in the best scores. For reconstruction errors, regarding calibrating on OpenR1Math/FineWeb and testing on FineWeb/OpenR1Math, see Figures 7 and 8.

Table 9: Ablation of the source of data. We consider `kvtc` calibrated fully on FineWeb (Penedo et al., 2024), fully on OpenMathR1 (R1, 2025) and a 50/50 mixture.

Method	Data	CR	GSM	MMLU	QASP	LITM	VT
Mistral-NeMo 12B							
$\texttt{kvtc}_{32\times}$	FineWeb + OpenR1Math	31-43	$62.2_{1.3}$	$63.8_{0.4}$	37.5	$99.6_{0.1}$	$98.7_{0.4}$
	FineWeb	31-43	$63.5_{1.3}$	$63.5_{0.4}$	37.5	$98.5_{0.2}$	$98.9_{0.3}$
	OpenR1Math	31-43	$63.5_{1.3}$	$64.8_{0.4}$	37.8	$99.7_{0.1}$	$99.3_{0.3}$
$\texttt{kvtc}_{64\times}$	FineWeb + OpenR1Math	51-87	$61.9_{1.3}$	$61.4_{0.4}$	38.0	$95.3_{0.3}$	$98.0_{0.4}$
	FineWeb	63-87	$63.2_{1.3}$	$59.9_{0.4}$	36.5	$92.0_{0.3}$	$97.4_{0.5}$
	OpenR1Math	63-86	$62.2_{1.3}$	$63.7_{0.4}$	38.2	$98.5_{0.2}$	$96.5_{0.6}$
$\texttt{kvtc}_{256\times}$	FineWeb + OpenR1Math	148-340	$60.0_{1.3}$	$52.2_{0.4}$	31.6	$40.0_{0.6}$	$84.3_{1.1}$
	FineWeb	148-343	$57.9_{1.4}$	$51.5_{0.4}$	31.0	$14.1_{0.4}$	$74.5_{1.4}$
	OpenR1Math	156-342	$56.6_{1.4}$	$54.0_{0.4}$	29.4	$21.0_{0.5}$	$81.3_{1.2}$

C.7 Sliding window size vs downstream performance

We ablate the influence of sliding window of recent not-compressed tokens on downstream performance in Table 10. We observe that increasing the length of sliding window increases the model performance, with most noticeable differences between sliding windows of lengths 1-16 and sliding windows of lengths 64-128.

Table 10: Ablation of the sliding window size of recent tokens that are not compressed.

Method	Window Size	CR	GSM	MMLU	QASP	LITM	VT
Mistral-NeMo 12B							
$\texttt{kvtc}_{64\times}$	1	61-87	$53.0_{1.4}$	$54.9_{0.4}$	36.4	$88.6_{0.4}$	$90.7_{0.9}$
	16	60-87	$60.1_{1.3}$	$56.4_{0.4}$	37.7	$88.9_{0.4}$	$95.8_{0.6}$
	64	59-87	$62.2_{1.3}$	$59.0_{0.4}$	37.8	$93.0_{0.3}$	$98.0_{0.4}$
	128	51-87	$61.9_{1.3}$	$61.4_{0.4}$	38.0	$95.3_{0.3}$	$98.0_{0.4}$

C.8 STANDARD ERROR OF THE MAIN RESULTS

In Table 11 we attach the results from Table 2 with their standard error, as bootstrapped by LM Evaluation Harness (where available) (Gao et al., 2024). We note that downstream evaluation runs, except the Qwen models, were performed using 1 seed and greedy evaluation.

Table 11: Downstream task results, presented also in Table 2, here shown with standard error as reported by LM Evaluation Harness (where available).

Method	GSM8K (Math & Knowledge)	MMLU (Math & Knowledge)	QASPER (Long Context)	LITM 100 (Long Context)	RULER-VT (Long Context)
Llama 3.1 8B					
vanilla	$56.8_{1.4}$	$60.5_{0.4}$	40.4	$99.4_{0.1}$	$99.8_{0.2}$
GEAR 2bit	$52.8_{1.4}$	$59.6_{0.4}$	40.4	$96.9_{0.2}$	$99.8_{0.2}$
KIVI 2bit	$52.8_{1.4}$	$59.6_{0.4}$	39.1	$88.8_{0.4}$	$98.9_{0.3}$
$\text{xKV}^{2/16\text{ 4key}}_{3/16\text{ 4value}}$	$56.6_{1.4}$	$59.5_{0.4}$	35.6	$99.9_{0.0}$	$99.8_{0.2}$
$\texttt{kvtc}_{8\times}$	$57.0_{1.4}$	$59.8_{0.4}$	40.1	$99.3_{0.1}$	$99.1_{0.3}$
$\texttt{kvtc}_{16\times}$	$56.9_{1.4}$	$60.1_{0.4}$	40.7	$99.3_{0.1}$	$99.1_{0.3}$
$\texttt{kvtc}_{32\times}$	$57.8_{1.4}$	$60.6_{0.4}$	39.4	$99.1_{0.1}$	$98.9_{0.3}$
$\texttt{kvtc}_{64\times}$	$57.2_{1.4}$	$60.7_{0.4}$	37.8	$90.2_{0.4}$	$95.9_{0.6}$
$\text{H}_2\text{O}^{1/16\text{ recent}}_{1/16\text{ past}}$	$54.3_{1.4}$	$44.3_{0.4}$	34.3	$20.2_{0.5}$	$50.4_{1.6}$
$\text{TOVA}\frac{1}{8}$	$54.5_{1.4}$	$44.8_{0.4}$	38.6	$1.2_{0.1}$	$99.7_{0.2}$
MN-Minitron 8B					
Vanilla	$59.1_{1.4}$	$64.3_{0.4}$	38.2	$99.8_{0.0}$	$99.4_{0.3}$
GEAR 2bit	$57.9_{1.4}$	$63.6_{0.4}$	38.2	$96.0_{0.2}$	$98.3_{0.4}$
KIVI 2bit	$58.0_{1.4}$	$63.2_{0.4}$	38.2	$86.3_{0.4}$	$96.8_{0.6}$
$\text{xKV}^{2/16\text{ 5key}}_{3/16\text{ 5value}}$	$59.3_{1.4}$	$63.1_{0.4}$	34.5	$99.6_{0.1}$	$99.1_{0.3}$
$\texttt{kvtc}_{8\times}$	$60.6_{1.3}$	$64.2_{0.4}$	39.1	$99.4_{0.1}$	$98.8_{0.3}$
$\texttt{kvtc}_{16\times}$	$60.3_{1.3}$	$64.1_{0.4}$	38.6	$99.3_{0.1}$	$98.8_{0.3}$
$\texttt{kvtc}_{32\times}$	$59.1_{1.4}$	$63.7_{0.4}$	37.7	$86.9_{0.4}$	$96.0_{0.6}$
$\texttt{kvtc}_{64\times}$	$57.8_{1.4}$	$62.1_{0.4}$	38.1	$59.5_{0.6}$	$93.4_{0.8}$
$\text{H}_2\text{O}^{1/16\text{ recent}}_{1/16\text{ past}}$	$55.3_{1.4}$	$43.5_{0.4}$	30.0	$16.6_{0.5}$	$39.2_{1.5}$
$\text{TOVA}\frac{1}{8}$	$59.2_{1.4}$	$48.1_{0.4}$	33.9	$0.3_{0.1}$	$99.3_{0.3}$
Mistral-NeMo 12B					
Vanilla	$61.9_{1.3}$	$64.5_{0.4}$	38.4	$99.5_{0.1}$	$99.8_{0.2}$
GEAR 2bit	$59.8_{1.4}$	$64.0_{0.4}$	38.6	$96.9_{0.2}$	$99.4_{0.3}$
KIVI 2bit	$59.7_{1.4}$	$64.3_{0.4}$	38.2	$91.9_{0.3}$	$98.3_{0.4}$
$\text{xKV}^{2/16\text{ 5key}}_{3/16\text{ 5value}}$	$61.9_{1.3}$	$63.9_{0.4}$	33.5	$97.9_{0.2}$	$99.4_{0.3}$
$\texttt{kvtc}_{8\times}$	$62.5_{1.3}$	$64.6_{0.4}$	37.6	$99.9_{0.0}$	$99.5_{0.2}$
$\texttt{kvtc}_{16\times}$	$62.0_{1.3}$	$64.4_{0.4}$	37.6	$99.8_{0.0}$	$99.5_{0.2}$
$\texttt{kvtc}_{32\times}$	$62.2_{1.3}$	$63.8_{0.4}$	37.5	$99.6_{0.1}$	$98.7_{0.4}$
$\texttt{kvtc}_{64\times}$	$61.9_{1.3}$	$61.4_{0.4}$	38.0	$95.3_{0.3}$	$98.0_{0.4}$
$\text{H}_2\text{O}^{1/16\text{ recent}}_{1/16\text{ past}}$	$57.0_{1.4}$	$45.4_{0.4}$	29.5	$16.2_{0.5}$	$35.2_{1.5}$
$\text{TOVA}\frac{1}{8}$	$60.3_{1.3}$	$49.0_{0.4}$	36.0	$8.7_{0.4}$	$99.6_{0.2}$

C.9 PCA MATRIX SIZES

In Table 12 we present the sizes of PCA projection matrices (V from $U\Sigma V^\top$) after being computed via (Halko et al., 2010) algorithm. We note that the sizes are only a relatively small fraction of the model parameters, and that they can be further reduced by the DP algorithm depending on the desired compression ratio.

Table 12: Number of parameters used by PCA matrices, before DP, for the tested models. We note that despite treating keys and values separately, while using feature capacity for the efficient (Halko et al., 2010) PCA calculation algorithm. For example, Llama 3.1 8B has 32 layers, each with 8 key/value heads, each head of size 128. Therefore, after cross-head concatenation, each key/value has $32 \times 8 \times 128 = 32768$ features. The PCA projection V is cut to the first 10K principal components by (Halko et al., 2010) algorithm for efficiency, resulting in $32768 \times 10000 \simeq 328M$ parameters. Further DP bit allocation can remove additional principal directions depending on the desired compression ratio. Both models and PCA projection matrices are stored in 16bit precision.

Model	Key/Value Features	Key/Value PCA Features Cap	Key/Value PCA Params
Qwen 2.5 R1 1.5B	$28 \times 2 \times 128 = 7168$	8K	51M
Qwen 2.5 R1 7B	$28 \times 4 \times 128 = 14336$		115M
Llama 3.1 8B	$32 \times 8 \times 128 = 32768$	10K	328M
Llama 3.3 70B	$80 \times 8 \times 128 = 81920$		819M
Mistral NeMo 12B	$40 \times 8 \times 128 = 40960$		410M

C.10 Dynamic programming algorithm

Below, we present the pseudocode for the dynamic programming precision assignment along with a proof sketch.

Dynamic Programming Precision Assignment Pseudocode

```
D                                # calibration data matrix of shape (batch, num_features)
m = D.mean(dim=0, keepdim=True)  # of each feature across the batch dimension

U, S, V = svd(D - m)             # D - m = U @ S @ V.T

P = U@S                          # assume columns sorted by singular values
batch, num_considered_features = P.shape

# initial_reconstruction_error corresponds to quantizing everything with zero bits
initial_reconstruction_error = (P*P).sum()    #  squared Frobenius norm

# set to initial_reconstruction_error as we assume that the data is initially quantized with 0 bits
# and we progressively consider non-zero quantization of more and more features
best_error = tensor(shape=(num_considered_features + 1, max_bit_budget + 1),
↪  values=initial_reconstruction_error)
best_error_type = array(shape=(num_considered_features + 1, max_bit_budget + 1), values=0)
best_error_block_size = tensor(shape=(num_considered_features + 1, max_bit_budget + 1), values=0)
best_error_bit_cost = tensor(shape=(num_considered_features + 1, max_bit_budget + 1), values=0)

# we assume that block sizes are > 0
allowed_block_sizes = [1, 16, 64, 256, 1024]

# We assume the presence of a None type that quantizes data to the array of zeros.
# We count bit usage of this type as 0,
# because it directly corresponds to the removal of principal components.
types = [None, int2, int4, fp8]

for i in range(1, num_considered_features + 1):
  for block_size in allowed_block_sizes:
    if block_size <= i:
      assert block_size > 0
      # the loop below can be parallelized
      for budget in range(1, max_bit_budget + 1):
        if best_error[i, budget] > best_error[i, budget - 1]:
          best_error[i, budget] = best_error[i, budget - 1]
          best_error_type[i, budget] = best_error_type[i, budget - 1]
          best_error_block_size[i, budget] = best_error_block_size[i, budget - 1]
          best_error_bit_cost[i, budget] = best_error_bit_cost[i, budget - 1]

        for t in types:

          block_to_quantize = P[:, i - block_size:i]

          quantized_data, used_bits = simulate_quantization(block_to_quantize, t)
          if used_bits <= budget:
            zero_bit_quantize_error = (block_to_quantize * block_to_quantize).sum()
            quantization_error = block_to_quantize - quantized_data
            quantization_error = (quantization_error * quantization_error).sum()

            error_change = -zero_bit_quantize_error + quantization_error

            if best_error[i, budget] > error_change + best_error[i - block_size, budget - used_bits]:
              best_error[i, budget] = error_change + best_error[i - block_size, budget - used_bits]
              best_error_type[i, budget] = t
              best_error_block_size[i, budget] = block_size
              best_error_bit_cost[i, budget] = used_bits

# we can use best_error, best_error_type, best_error_block_size and best_error_bit_cost tables to get
↪  the quantization for a given budget
```

The proof of the optimality follows by simple induction on `i` and `budget`. To be more precise we want to prove that `best_error[j, q]` is the smallest reconstruction error (squared Frobenius norm) one can achieve when considering first j features of P (setting other features to 0 – 0-bit quantization) and quantization restricted to types from `types` that can only be used to quantize blocks of contiguous features of sizes in `allowed_block_sizes` sizes, while utilizing no more than `budget` bits. For simplicity we assume that the smallest reconstruction error (squared Frobenius norm) for q=0 budget cases is `initial_reconstruction_error = (P*P).sum()`. Then the proof by induction can be conducted as follows:

- For `i=0` or `budget=0` we have that if we consider quantization of the first 0 features and leave other features as zeros or budget of size 0, then the reconstruction error is indeed `(P*P).sum()`.
- Then to prove for `i>0` and `budget>0` we assume the optimality of `best_error[j, q]` for `j<i`, and for `j=i` and `q<budget`. Then we note that the algorithm enumerates all

possible quantization blocks that the quantization of the first `i` features can end with within the budget `budget`.

Computational complexity can be directly inferred from the pseudo-code:

$$\begin{aligned} \mathcal{O}(\text{num\_considered\_features} \times \\ |\text{allowed\_block\_sizes}| \times \\ \text{max\_bit\_budget} \times \\ |\text{types}| \times \\ q_{\text{sim}}(\max\{\text{allowed\_block\_sizes}\}, \text{batch})) \end{aligned}$$

where

$$q_{\text{sim}}(\max\{\text{allowed\_block\_sizes}\}, \text{batch})$$

is the time taken to simulate quantization. Assuming that $|\text{allowed\_block\_sizes}|$ and $|\text{types}|$ are constant and quantization simulation can be performed in

$$\mathcal{O}(\max\{\text{allowed\_block\_sizes}\} \times \text{batch})$$

we can write the asymptotic bound on the algorithm runtime as:

$$\mathcal{O}(\text{num\_considered\_features} \times \text{max\_bit\_budget} \times \text{batch})$$

We additionally provide the runtime of the algorithm in Table 8 in Appendix C.5.