# ORTHOKV: EXTREME KV CACHE COMPRESSION WITH EXACT ORTHOGONAL SPARSE CODING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

The KV cache is a major memory bottleneck in long-context inference. Sparse coding is attractive for extreme KV cache compression, but existing approaches rely on iterative sparse approximation over large overcomplete dictionaries, introducing substantial encoding overhead. We introduce ORTHOKV, which replaces layer-wise overcomplete dictionaries with complete orthogonal dictionaries specialized to individual KV heads. Per-head specialization helps compensate for the reduced representational capacity of complete dictionaries, while orthogonality makes sparse coding exact and non-iterative. The best $k$-term approximation under squared reconstruction error is obtained by a single transform followed by selecting the $k$ coefficients with the largest absolute values. The same result extends across heads. With a total budget of $B$ coefficients, one simply keeps the $B$ largest coefficients in absolute value across all heads. We learn the dictionaries with a sparsity-promoting objective that does not depend on the target sparsity, so the same dictionaries can be used at different compression ratios without retraining. Complete dictionaries also enable smaller coefficient indices and more compact sparse codes. In the extreme compression regime, ORTHOKV uses less KV cache memory than Lexico while matching or improving its quality in most evaluated settings and substantially reduces sparse coding latency. Compared with full KV caching, ORTHOKV achieves higher throughput at larger batch sizes and supports batches that would otherwise run out of memory.

## 1 INTRODUCTION

Autoregressive large language models (LLMs) repeatedly attend to the keys and values of all preceding tokens. Storing these tensors in the key-value (KV) cache avoids recomputation, but its memory consumption grows linearly with sequence length, batch size, and model depth. For long-context inference, the KV cache becomes a major bottleneck to efficient model serving. This has motivated a broad range of KV cache compression methods based on quantization (Liu et al., 2024b; Hooper et al., 2024), token eviction (Zhang et al., 2023; Li et al., 2024), low-rank approximation (Chang et al., 2025; Lin et al., 2025), and sparse representations (Zhang et al., 2025; Kim et al., 2025).

Sparse coding is particularly attractive in the extreme compression regime because it retains a representation for every cached token while assigning storage only to a small number of active coefficients. CSR (Zhang et al., 2025) represents KV vectors using sparse coefficients and indices over a learned dictionary. Lexico (Kim et al., 2025) uses an overcomplete dictionary for each layer and encodes KV vectors using orthogonal matching pursuit (OMP) (Pati et al., 1993). However, combining large overcomplete dictionaries with iterative OMP introduces substantial computational overhead. Encoding requires multiple sequential searches over thousands of atoms, inference may require large precomputed Gram matrices, and the released training pipeline learns separate dictionaries for different sparsity levels. These costs are especially problematic during autoregressive decoding, where new keys and values must be encoded at every layer and generation step.

Overcomplete dictionaries are widely used in sparse coding. In this work, we ask whether overcompleteness is necessary for sparse coding-based KV cache compression. Prior work has shown that attention heads exhibit different attention patterns, roles, and sensitivity to KV cache compression (Feng et al., 2025; Fu et al., 2025; Xiao et al., 2025). Motivated by this head-wise heterogeneity, we replace layer-wise overcomplete dictionaries with complete orthogonal dictionaries specialized



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

to individual KV heads. While restricting the dictionary to a complete orthogonal basis reduces its representational capacity, per-head specialization allows each dictionary to model a more specific KV distribution rather than sharing one dictionary across all heads in a layer. Specifically, OrthoKV learns complete orthogonal dictionaries $\boldsymbol{D}^K_{\ell,h}, \boldsymbol{D}^V_{\ell,h} \in \mathrm{O}(d)$ for every layer $\ell$ and KV head $h$, where $\mathrm{O}(d)$ denotes the set of $d \times d$ orthogonal matrices. For an orthogonal dictionary, the exact best $k$-term approximation under squared reconstruction error is obtained by transforming the KV vector and retaining the $k$ coefficients with the largest absolute values. This eliminates the iterative support selection and least-squares updates required by OMP.

Orthogonality also allows the number of retained coefficients to vary across heads. Rather than assigning the same sparsity to every head, OrthoKV uses a total budget of $B$ coefficients for each layer. The exact solution is obtained by retaining the $B$ coefficients with the largest absolute values across all heads, which automatically determines how many coefficients are assigned to each head. For the models considered in this work, each KV head has dimension $d = 128$, so both coefficient indices and the number of coefficients assigned to each head can be stored in one byte. This further reduces the storage overhead of the sparse representation compared with overcomplete dictionaries.

We learn each orthogonal dictionary using a sparsity-promoting objective based on $\ell_p$-norm maximization over the orthogonal group (Zhai et al., 2020; Shen et al., 2020). The objective encourages the transformed KV vectors to have a small number of coefficients with large absolute values, making them more suitable for sparse approximation. Importantly, this objective does not depend on the target sparsity $k$. As a result, the same learned dictionaries can be used across different compression ratios without retraining. This differs from Lexico, whose open-source pipeline trains separate dictionaries for different sparsity levels and uses each dictionary at its corresponding sparsity during inference.

We evaluate OrthoKV on reasoning and long-context benchmarks under extreme KV cache compression. Across the evaluated compression budgets, OrthoKV uses less KV cache memory than Lexico while achieving comparable or better downstream quality in most settings. At the same time, replacing iterative OMP with orthogonal transforms and coefficient selection substantially reduces sparse coding latency. Compared with full KV caching, OrthoKV achieves higher throughput at larger batch sizes and accommodates larger batches within the same GPU memory budget. These results show that complete orthogonal dictionaries enable effective sparse KV cache compression with practical benefits for autoregressive inference.

In summary, our contributions are:

- We introduce per-head complete orthogonal dictionaries for KV cache compression, enabling exact sparse coding through an orthogonal transform and top-$k$ selection instead of iterative OMP.
- We derive an optimal allocation of a fixed coefficient budget across KV heads and reduce sparse storage overhead through one-byte atom indices and per-head coefficient counts.
- We learn dictionaries that support multiple compression budgets without retraining. Experiments demonstrate competitive quality with smaller KV caches and substantially shorter evaluation time than Lexico. Compared with full KV caching, OrthoKV achieves higher throughput at larger batch sizes and supports larger batches within the same GPU memory budget.

## 2 OrthoKV

### 2.1 Background

Sparse coding approximates a vector $\boldsymbol{y} \in \mathbb{R}^d$ using a small number of atoms from a dictionary $\boldsymbol{D} \in \mathbb{R}^{d \times N}$:

$$\begin{aligned} \min_{\boldsymbol{a} \in \mathbb{R}^N} \quad & \|\boldsymbol{y} - \boldsymbol{D}\boldsymbol{a}\|_2^2 \\ \text{s.t.} \quad & \|\boldsymbol{a}\|_0 \le s, \end{aligned} \tag{1}$$

where the columns of $D$ are dictionary atoms and $s$ is the coefficient budget. For KV cache compression, sparse coefficients and their atom indices are stored instead of dense KV vectors, while the dictionary is shared across tokens.

CSR (Zhang et al., 2025) combines an offline neural dictionary with an online dictionary sampled from the KV cache during inference. It uses matching pursuit (MP) (Mallat & Zhang, 1993) to encode KV vectors, iteratively selecting atoms and updating the residual. However, online dictionary construction and iterative sparse coding can incur substantial overhead during inference.

Lexico (Kim et al., 2025) learns overcomplete dictionaries for keys and values in each layer. Unlike CSR, Lexico uses OMP to encode KV vectors, which iteratively selects an atom and refits the coefficients over all selected atoms. Repeated atom searches and coefficient fitting incur substantial encoding overhead during inference. This motivates investigating whether complete orthogonal dictionaries can provide effective KV cache compression without iterative sparse coding.

### 2.2 Per-Head Orthogonal Sparse Coding

For each layer $\ell$ and KV head $h$, OrthoKV uses separate complete orthogonal dictionaries for keys and values,

$$\boldsymbol{D}^K_{\ell,h}, \boldsymbol{D}^V_{\ell,h} \in \mathrm{O}(d), \tag{2}$$

where $d$ is the dimension of each KV head. For a KV vector $\boldsymbol{y} \in \mathbb{R}^d$ and its corresponding dictionary $\boldsymbol{D}$, we solve the sparse approximation problem

$$\min_{\|\boldsymbol{a}\|_0 \le k} \|\boldsymbol{y} - \boldsymbol{D}\boldsymbol{a}\|_2^2. \tag{3}$$

**Proposition 1.** *Let $\boldsymbol{D} \in \mathrm{O}(d)$. An optimal solution to problem (3) is given by*

$$\boldsymbol{a}^* = H_k(\boldsymbol{D}^\mathsf{T}\boldsymbol{y}), \tag{4}$$

*where $H_k(\boldsymbol{D}^\mathsf{T}\boldsymbol{y})$ retains the $k$ entries of $\boldsymbol{D}^\mathsf{T}\boldsymbol{y}$ with the largest absolute values.*

*Proof.* Since $\boldsymbol{D}$ is orthogonal,

$$\|\boldsymbol{y} - \boldsymbol{D}\boldsymbol{a}\|_2^2 = \|\boldsymbol{D}^\mathsf{T}\boldsymbol{y} - \boldsymbol{a}\|_2^2. \tag{5}$$

Under the constraint $\|\boldsymbol{a}\|_0 \le k$, this error is minimized by retaining the $k$ entries of $\boldsymbol{D}^\mathsf{T}\boldsymbol{y}$ with the largest absolute values and setting the rest to zero. □

Therefore, sparse coding in OrthoKV requires only an orthogonal transform followed by coefficient selection. Unlike OMP in Lexico, OrthoKV does not require iterative support selection or repeated least-squares updates.

### 2.3 Coefficient Allocation Across Heads

Prior work has shown that KV heads can differ substantially in their attention behavior and sensitivity to cache compression (Feng et al., 2025; Fu et al., 2025). This suggests that assigning the same number of coefficients to every head may be unnecessarily restrictive. Instead, for each token, OrthoKV allocates a total budget of $B = H_{\mathrm{kv}}k$ coefficients across the $H_{\mathrm{kv}}$ heads within each layer, separately for keys and values, where $k$ is the average coefficient budget per head. For notational simplicity, we omit the layer and token indices. The coefficient allocation is formulated as

$$\begin{aligned} \min_{\{\boldsymbol{a}_h\}_{h=1}^{H_{\mathrm{kv}}}} \quad & \sum_{h=1}^{H_{\mathrm{kv}}} \|\boldsymbol{y}_h - \boldsymbol{D}_h\boldsymbol{a}_h\|_2^2 \\ \text{s.t.} \quad & \sum_{h=1}^{H_{\mathrm{kv}}} \|\boldsymbol{a}_h\|_0 \le B. \end{aligned} \tag{6}$$

**Proposition 2.** *Let $\boldsymbol{D}_h \in \mathrm{O}(d)$ and $\boldsymbol{z}_h = \boldsymbol{D}_h^\mathsf{T}\boldsymbol{y}_h$ for each head $h$. An optimal solution to problem (6) is obtained by retaining the $B$ entries with the largest absolute values among $\{\boldsymbol{z}_h\}_{h=1}^{H_{\mathrm{kv}}}$ and setting all remaining entries to zero.*

*Proof.* Since each $\boldsymbol{D}_h$ is orthogonal,

$$\sum_{h=1}^{H_{\mathrm{kv}}} \|\boldsymbol{y}_h - \boldsymbol{D}_h \boldsymbol{a}_h\|_2^2 = \sum_{h=1}^{H_{\mathrm{kv}}} \|\boldsymbol{z}_h - \boldsymbol{a}_h\|_2^2. \tag{7}$$

For any retained entry $z_{h,j}$, setting $a_{h,j} = z_{h,j}$ removes $|z_{h,j}|^2$ from the reconstruction error. Therefore, under a total budget of $B$ nonzero coefficients, the error is minimized by retaining the $B$ entries with the largest absolute values across all heads. □

Therefore, the number of retained coefficients for each head is determined by the transformed KV vectors themselves rather than fixed in advance. If $s_h$ denotes the number assigned to head $h$, then

$$\sum_{h=1}^{H_{\mathrm{kv}}} s_h = B. \tag{8}$$

Heads with more large coefficients receive more of the shared budget, while the total number of retained coefficients remains fixed. In our experiments, we also observe that this allocation improves downstream quality over uniform allocation (see Section 3 for details).

### 2.4 Dictionary Learning

Following Lexico (Kim et al., 2025), we use WikiText-103 (Merity et al., 2017) to collect KV vectors for dictionary learning. Our dictionary learning differs from Lexico in two key aspects. First, we learn separate complete orthogonal dictionaries for individual KV heads, rather than a layer-wise overcomplete dictionary. Second, instead of alternating between sparse coding at a specified sparsity level and dictionary updates, we directly optimize a sparsity-promoting objective over the set of orthogonal matrices. We describe the learning problem for a single dictionary (key or value, at a fixed layer and head) and denote its training samples by $\{\boldsymbol{y}_i\}_{i=1}^n$. Following Shen et al. (2020), we learn the dictionary by maximizing the $\ell_3$-based objective over the set of orthogonal matrices:

$$\max_{\boldsymbol{D} \in \mathrm{O}(d)} f(\boldsymbol{D}) = \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{D}^{\mathsf{T}} \boldsymbol{y}_i\|_3^3. \tag{9}$$

Because orthogonal transforms preserve the $\ell_2$-norm of each sample, maximizing the $\ell_3$-based objective encourages a small number of coefficients to have large absolute values and the remaining coefficients to have smaller absolute values. This makes the transformed vectors more suitable for sparse approximation. Importantly, problem (9) does not depend on the sparsity level used at inference. Therefore, the same learned dictionary can be used with different numbers of retained coefficients without retraining.

We optimize problem (9) using the generalized power method proposed by Journée et al. (2010) and adopted for $\ell_3$-based complete dictionary learning by Shen et al. (2020). Let $\boldsymbol{z}_i = \boldsymbol{D}^{\mathsf{T}} \boldsymbol{y}_i$. The gradient of $f$ is

$$\boldsymbol{G} = \frac{3}{n} \sum_{i=1}^{n} \boldsymbol{y}_i (\boldsymbol{z}_i \odot |\boldsymbol{z}_i|)^{\mathsf{T}}, \tag{10}$$

where $\odot$ denotes the Hadamard product. Given the singular value decomposition $\boldsymbol{G} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}}$, the dictionary is updated as

$$\boldsymbol{D} \leftarrow \boldsymbol{U}\boldsymbol{V}^{\mathsf{T}}, \tag{11}$$

which is the orthogonal matrix that maximizes the linear subproblem in the generalized power method. Algorithm 1 summarizes the learning procedure.

We learn each per-head dictionary from 50,000 KV vectors using full-batch generalized power updates. To reduce memory usage, we accumulate gradients over chunks of 8,192 vectors before each update. We use 200 updates for each dictionary. Despite learning separate dictionaries for individual KV heads, our dictionary learning is fast in practice. Table 1 compares the total training time with Lexico on a single NVIDIA A100 80GB GPU. All per-head dictionaries in OrthoKV are trained sequentially, and a single set of learned dictionaries supports all compression budgets.

Therefore, OrthoKV requires substantially less offline dictionary learning time and, unlike Lexico, does not require separate training runs for different sparsity levels.

**Algorithm 1** Per-Head Orthogonal Dictionary Learning

**Require:** Training vectors $\{\boldsymbol{y}_i\}_{i=1}^{n}$ for one layer, KV head, and key/value; number of updates $T$; chunk size $C$

Initialize $\boldsymbol{D}^{(0)} \in \mathrm{O}(d)$ randomly
Partition $\{1, \dots, n\}$ into chunks $\mathcal{I}_1, \dots, \mathcal{I}_M$ of size at most $C$
**for** $t = 0, \dots, T-1$ **do**
  $\boldsymbol{G} \leftarrow 0$
  **for** $m = 1, \dots, M$ **do**
    $\boldsymbol{Y}_m \leftarrow [\boldsymbol{y}_i]_{i \in \mathcal{I}_m}$ ▷ Samples are columns
    $\boldsymbol{Z}_m \leftarrow (\boldsymbol{D}^{(t)})^{\mathsf{T}} \boldsymbol{Y}_m$
    $\boldsymbol{G} \leftarrow \boldsymbol{G} + \frac{3}{n} \boldsymbol{Y}_m (\boldsymbol{Z}_m \odot |\boldsymbol{Z}_m|)^{\mathsf{T}}$
  **end for**
  $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}} \leftarrow \mathrm{SVD}(\boldsymbol{G})$
  $\boldsymbol{D}^{(t+1)} \leftarrow \boldsymbol{U}\boldsymbol{V}^{\mathsf{T}}$
**end for**
**return** $\boldsymbol{D}^{(T)}$

Table 1: Dictionary training time in minutes. A single set of OrthoKV dictionaries supports all compression budgets. All per-head dictionaries are trained sequentially.

| Method | Llama-3.1-8B-Instruct | Mistral-7B-Instruct-v0.3 |
|---|---|---|
| OrthoKV | 13.8 | 13.4 |
| Lexico ($s = 4, N = 4096$) | 46.1 | 45.9 |
| Lexico ($s = 8, N = 4096$) | 79.9 | 79.8 |
| Lexico ($s = 12, N = 4096$) | 114.6 | 115.2 |

### 2.5 Storage and Inference

Following Lexico (Kim et al., 2025), we maintain a full-precision buffer containing the most recent $n_b = 128$ tokens. Prefill attention uses uncompressed KV vectors. Before decoding begins, we compress the prompt prefix while retaining the most recent $n_b$ tokens in full precision. During decoding, each new token enters the buffer, and the oldest token is removed from the buffer and compressed for use in subsequent attention steps. Attention combines the compressed prefix with the full-precision buffer under a shared softmax normalization.

For a given KV head, let $\boldsymbol{K} \approx \boldsymbol{Z}_K \boldsymbol{D}_K^{\mathsf{T}}$ and $\boldsymbol{V} \approx \boldsymbol{Z}_V \boldsymbol{D}_V^{\mathsf{T}}$, where $\boldsymbol{Z}_K$ and $\boldsymbol{Z}_V$ contain the sparse coefficients. The attention computation can be expressed as

$$\boldsymbol{Q}\boldsymbol{K}^{\mathsf{T}} \approx (\boldsymbol{Q}\boldsymbol{D}_K)\boldsymbol{Z}_K^{\mathsf{T}}, \tag{12}$$

$$\boldsymbol{A}\boldsymbol{V} \approx (\boldsymbol{A}\boldsymbol{Z}_V)\boldsymbol{D}_V^{\mathsf{T}}, \tag{13}$$

where $\boldsymbol{A}$ denotes the attention weights for the compressed prefix, obtained by jointly normalizing its logits with those of the full-precision buffer.

We next present the storage format of the compressed KV vectors. For the models considered in this work, each KV head has dimension $d = 128$, so each coefficient index and the per-head coefficient count $s_h$ can be stored in one byte. For a head assigned $s_h$ coefficients, we store $s_h$ FP8 coefficient values, $s_h$ one-byte indices, and one byte for $s_h$, for a total of $2s_h + 1$ bytes. Since $\sum_{h=1}^{H_{\mathrm{kv}}} s_h = B$, the compressed representation of one token at one layer requires $2B + H_{\mathrm{kv}}$ bytes for either keys or values.

In contrast, for each compressed KV vector corresponding to one head, the released Lexico implementation stores $s$ FP8 coefficients, $s$ two-byte atom indices, and four bytes of row metadata, resulting in $3s + 4$ bytes[1]. With $H_{\mathrm{kv}}$ heads, this corresponds to $H_{\mathrm{kv}}(3s + 4)$ bytes per token and layer for either keys or values. This gives OrthoKV a more compact sparse representation than Lexico, since each coefficient index and each per-head allocation can be stored in one byte.

[1]The Lexico paper reports $3s+2$ bytes using a two-byte row offset. We follow the released implementation, which uses four-byte row metadata, giving $3s + 4$ bytes.

Beyond the compressed KV cache itself, the dictionaries introduce a fixed memory cost. For the models evaluated in this work, which have 32 layers, $H_{\text{kv}} = 8$, and $d = 128$, all key and value dictionaries in OrthoKV require 16 MiB in FP16, compared with 64 MiB for Lexico with $N = 4096$. To accelerate OMP in Lexico, the dictionary Gram matrices can be precomputed and stored. For Lexico with $N = 4096$, storing the Gram matrices for all key and value dictionaries requires about 2 GiB in FP16. OrthoKV does not require this additional storage.

# 3 Experiments

## 3.1 Experimental Setup

**Models and evaluation protocol.** We evaluate OrthoKV on Llama-3.1-8B-Instruct and Mistral-7B-Instruct-v0.3 using GSM8K (Cobbe et al., 2021) and LongBench (Bai et al., 2024). We compare against the full FP16 KV cache, KIVI (Liu et al., 2024b), and Lexico (Kim et al., 2025). We focus on methods that retain a representation for every cached token, rather than token eviction methods that reduce memory by discarding selected tokens.

For GSM8K, we evaluate the full test set with 5-shot prompting and report accuracy. For LongBench, we evaluate on the same eight tasks (Qasper (Dasigi et al., 2021), QMSum (Zhong et al., 2021), MultiNews (Fabbri et al., 2019), TREC (Li & Roth, 2002), TriviaQA (Joshi et al., 2017), SAMSum (Gliwa et al., 2019), LCC (Guo et al., 2023), and RepoBench-P (Liu et al., 2024a)) used by Kim et al. (2025). We evaluate all examples in each task using its standard evaluation metric and report the macro average score across the eight tasks. For LongBench inputs longer than 8,192 tokens, we retain the first and last 4,096 tokens. We apply each model's chat template to GSM8K, Qasper, QMSum, and MultiNews. TREC, TriviaQA, SAMSum, LCC, and RepoBench-P use task prompts without a chat template. Task prompts are provided in Appendix A.

All experiments are conducted on a single NVIDIA A100 80GB GPU.

**Compression configurations.** For Lexico, we use dictionaries with $N = 4096$ atoms and sparsity levels $s \in \{4, 8, 12\}$. For OrthoKV, we use average coefficient budgets $k \in \{6, 12, 18\}$ per KV head with adaptive allocation across heads. These settings provide comparable storage budgets, with OrthoKV using slightly less KV cache memory due to its more compact sparse representation.

Both Lexico and OrthoKV store sparse coefficients in FP8 and maintain a full-precision buffer of the most recent 128 tokens. We precompute the dictionary Gram matrices to accelerate OMP in Lexico. For KIVI, both keys and values are quantized to 2 bits. We use a group size of 32 and a residual buffer of 128 tokens on LongBench, and a group size of 64 with a residual buffer of 64 tokens on GSM8K.

Reported KV cache sizes include coefficients or quantized values, indices, cache metadata, and the full-precision buffer, but exclude shared dictionaries and runtime workspace. Peak GPU memory is reported separately in the batch-size scaling experiment.

## 3.2 Quality under Extreme Compression

**LongBench results.** Table 2 reports the quality results on LongBench. We choose coefficient budgets so that OrthoKV is compared against Lexico at approximately matched total KV cache memory while remaining slightly smaller in every pair. On Llama-3.1-8B-Instruct, OrthoKV improves the average score at the two lower budgets and matches Lexico at the highest budget, while using approximately 1.1 percentage points less KV cache in each case. On Mistral-7B-Instruct-v0.3, OrthoKV consistently outperforms Lexico on the macro average across all three budgets, while also using less KV cache memory.

**GSM8K results.** Table 3 reports the quality results on GSM8K. Similarly, OrthoKV uses less KV cache memory than Lexico at all matched compression levels. On Llama-3.1-8B-Instruct, it improves accuracy at the two lower budgets, while Lexico performs better at the highest budget. On Mistral-7B-Instruct-v0.3, the two methods are comparable at the lowest budget, while OrthoKV achieves higher accuracy at the two larger budgets. The KV cache ratios on GSM8K are higher

Table 2: LongBench results. KV cache size is computed using the actual generated sequence length, including the full-precision buffer, and is reported relative to the FP16 KV cache for the same sequence length. For ORTHOKV, $k$ denotes the average number of retained coefficients per KV head.

| Method | KV Size | Qasper | QMSum | MultiNews | TREC | TriviaQA | SAMSum | LCC | RepoBench-P | Average |
|---|---|---|---|---|---|---|---|---|---|---|
| **Llama-3.1-8B-Instruct** | | | | | | | | | | |
| Full KV | 100.00% | 45.00 | 23.34 | 27.21 | 69.50 | 91.20 | 44.09 | 63.46 | 55.52 | 52.42 |
| KIVI-2 | 20.30% | 43.49 | 23.29 | 26.89 | 69.00 | 91.77 | 43.84 | 62.13 | 53.65 | 51.76 |
| Lexico $s = 4$ | 8.63% | 20.93 | 20.46 | 15.05 | 41.50 | 80.43 | 38.10 | 53.38 | 46.27 | 39.52 |
| ORTHOKV $k = 6$ | **7.49%** | 21.15 | 20.75 | 16.43 | 41.00 | 81.81 | 37.63 | 55.15 | 46.92 | **40.11** |
| Lexico $s = 8$ | 13.20% | 22.45 | 21.14 | 17.42 | 41.00 | 83.55 | 39.36 | 56.70 | 48.50 | 41.27 |
| ORTHOKV $k = 12$ | **12.05%** | 24.07 | 21.45 | 18.51 | 42.00 | 86.60 | 39.08 | 56.86 | 49.12 | **42.21** |
| Lexico $s = 12$ | 17.76% | 33.47 | 21.63 | 21.31 | 47.50 | 87.70 | 40.12 | 59.79 | 50.82 | **45.29** |
| ORTHOKV $k = 18$ | **16.62%** | 32.32 | 21.81 | 21.71 | 48.50 | 89.36 | 39.67 | 58.59 | 50.04 | 45.25 |
| **Mistral-7B-Instruct-v0.3** | | | | | | | | | | |
| Full KV | 100.00% | 37.49 | 23.98 | 26.56 | 74.00 | 88.39 | 47.59 | 58.34 | 57.06 | 51.68 |
| KIVI-2 | 20.09% | 37.26 | 23.70 | 26.25 | 74.00 | 88.34 | 48.53 | 57.55 | 55.95 | 51.45 |
| Lexico $s = 4$ | 8.40% | 21.73 | 20.57 | 16.25 | 44.00 | 83.00 | 40.87 | 49.09 | 47.32 | 40.35 |
| ORTHOKV $k = 6$ | **7.26%** | 21.29 | 20.18 | 17.79 | 46.50 | 82.72 | 41.95 | 51.36 | 47.88 | **41.21** |
| Lexico $s = 8$ | 12.98% | 24.39 | 20.78 | 18.97 | 47.00 | 86.73 | 42.61 | 53.65 | 50.35 | 43.06 |
| ORTHOKV $k = 12$ | **11.84%** | 27.25 | 21.10 | 21.18 | 58.00 | 85.42 | 42.62 | 52.93 | 51.34 | **44.98** |
| Lexico $s = 12$ | 17.57% | 32.51 | 22.57 | 22.29 | 68.00 | 88.27 | 44.67 | 54.62 | 53.96 | 48.36 |
| ORTHOKV $k = 18$ | **16.42%** | 32.15 | 22.06 | 23.33 | 72.00 | 87.32 | 44.10 | 55.53 | 54.81 | **48.91** |

Table 3: GSM8K results. We report the KV cache size relative to the full FP16 KV cache and accuracy.

| | **Llama-3.1-8B-Instruct** | | **Mistral-7B-Instruct-v0.3** | |
|---|---|---|---|---|
| Method | KV Size | Accuracy | KV Size | Accuracy |
| Full KV | 100.00% | 78.17 | 100.00% | 50.64 |
| KIVI-2 | 19.38% | 71.19 | 18.90% | 41.24 |
| Lexico $s = 4$ | 17.27% | 42.53 | 16.13% | **31.24** |
| ORTHOKV $k = 6$ | **16.27%** | **45.41** | **15.00%** | 31.16 |
| Lexico $s = 8$ | 21.49% | 50.42 | 20.27% | 35.48 |
| ORTHOKV $k = 12$ | **20.43%** | **53.90** | **19.20%** | **37.07** |
| Lexico $s = 12$ | 25.68% | **64.06** | 24.47% | 40.71 |
| ORTHOKV $k = 18$ | **24.61%** | 61.18 | **23.39%** | **41.55** |

than those on LongBench mainly because GSM8K sequences are shorter. As a result, the fixed full-precision buffer accounts for a larger fraction of the total KV cache.

### 3.3 INFERENCE EFFICIENCY

**End-to-end evaluation time at batch size 1.** Table 4 reports the per-token end-to-end evaluation time on LongBench at batch size 1, measured as the total evaluation wall-clock time divided by the total number of generated tokens across the eight tasks. ORTHOKV substantially reduces the per-token evaluation time compared with Lexico across all compression levels. On Llama-3.1-8B-Instruct, ORTHOKV requires 58.67–59.55 ms/token, compared with 604.97–788.66 ms/token for Lexico; on Mistral-7B-Instruct-v0.3, it requires 63.46–63.82 ms/token, compared with 713.42–925.79 ms/token. Overall, this corresponds to a $10.2\times$–$14.5\times$ speedup over Lexico. Moreover, the per-token evaluation time of ORTHOKV remains nearly constant as $k$ increases, whereas Lexico becomes progressively slower as $s$ increases. Full KV, Lexico, and ORTHOKV use eager attention during prefill, whereas KIVI uses FlashAttention-2 (Dao, 2024), which generally favors KIVI in

Table 4: End-to-end evaluation time per generated token on LongBench at batch size 1 (ms/token), measured as total evaluation wall time divided by the total number of generated tokens across the eight tasks. Full KV uses eager attention, while Lexico and OrthoKV use eager prefill attention and their respective compressed decode implementations. KIVI uses its FlashAttention-2-based implementation.

| Method | **Llama-3.1-8B-Instruct** | **Mistral-7B-Instruct-v0.3** |
|---|---|---|
| Full KV | 46.88 | 49.16 |
| KIVI-2 | 59.88 | 69.71 |
| Lexico $s = 4$ | 604.97 | 713.42 |
| OrthoKV $k = 6$ | **59.28** | **63.46** |
| Lexico $s = 8$ | 688.20 | 837.62 |
| OrthoKV $k = 12$ | **58.67** | **63.56** |
| Lexico $s = 12$ | 788.66 | 925.79 |
| OrthoKV $k = 18$ | **59.55** | **63.82** |

Table 5: Batch-size scaling on Llama-3.1-8B-Instruct. Columns labeled $k = 18$ and $k = 6$ correspond to OrthoKV, where $k$ is the average coefficient budget per KV head. Throughput is aggregated over the entire batch in tokens/s; peak memory is PyTorch peak allocated memory in GiB. Full KV runs out of memory (OOM) during prefill at batch size 32.

| | Overall throughput | | | Decode throughput | | | Peak memory | | |
|---|---|---|---|---|---|---|---|---|---|
| Batch | Full KV | $k = 18$ | $k = 6$ | Full KV | $k = 18$ | $k = 6$ | Full KV | $k = 18$ | $k = 6$ |
| 2 | **37.68** | 35.14 | 34.24 | 46.97 | **48.94** | 47.70 | 18.78 | 17.68 | **17.49** |
| 4 | 40.97 | 46.34 | **53.10** | 52.46 | 74.72 | **96.24** | 22.59 | 20.12 | **19.75** |
| 8 | 46.66 | 53.04 | **64.49** | 62.15 | 93.88 | **139.95** | 30.22 | 24.93 | **24.21** |
| 16 | 50.22 | 56.93 | **70.09** | 68.75 | 107.07 | **170.29** | 45.47 | 34.61 | **33.11** |
| 32 | OOM | 58.89 | **72.60** | OOM | 115.42 | **191.01** | OOM | 53.98 | **50.97** |

latency comparisons. Despite using eager attention during prefill, OrthoKV achieves per-token end-to-end evaluation time comparable to or lower than KIVI in these experiments.

**Scaling with batch size.** We further evaluate how OrthoKV scales with batch size on Llama-3.1-8B-Instruct using a single NVIDIA A100 80GB GPU. Each request contains 8,192 input tokens and generates 128 tokens with greedy decoding. We evaluate batch sizes of 2, 4, 8, 16, and 32. Both Full KV and OrthoKV use PyTorch scaled dot-product attention (SDPA) during prefill. Both methods use a 72 GiB PyTorch allocator budget. After one warmup run, we report the median of three runs using synchronized wall-clock timing.

Table 5 shows that the throughput benefits of OrthoKV become more pronounced at larger batch sizes. At batch size 2, both coefficient budgets yield slightly lower overall generation throughput than Full KV. At batch sizes 4–16, however, OrthoKV improves overall throughput by 13.1–13.7% with $k = 18$ and 29.6–39.6% with $k = 6$. The gains are larger during decoding: at batch size 16, decode throughput reaches 1.56× that of Full KV with $k = 18$ and 2.48× with $k = 6$. These results show that more aggressive compression can translate into additional decoding efficiency.

Both budgets also reduce peak allocated memory. At batch size 16, OrthoKV uses 34.61 GiB with $k = 18$ and 33.11 GiB with $k = 6$, compared with 45.47 GiB for Full KV, corresponding to reductions of 23.9% and 27.2%, respectively. Under the 72 GiB allocator budget, Full KV runs out of memory during prefill at batch size 32, whereas neither OrthoKV configuration does.

### 3.4 Ablation Studies

We conduct ablation studies on Llama-3.1-8B-Instruct using Qasper and QMSum, covering long-document question answering and query-focused meeting summarization, respectively. All comparisons use an average budget of $k = 12$ coefficients per KV head.

**Effect of per-head specialization.** We compare per-head dictionaries with a single dictionary shared across KV heads within each layer, keeping adaptive coefficient allocation in both settings. Both settings use the same collection of training vectors: each per-head dictionary uses 50,000 vectors, while the per-layer dictionary uses all 400,000 vectors across the eight KV heads. Per-head dictionaries improve Qasper from 23.36 to 24.07 and QMSum from 20.97 to 21.45, supporting the benefit of learning separate dictionaries for different heads.

**Effect of coefficient allocation across heads.** We compare uniform and adaptive allocation using the same per-head dictionaries. Uniform allocation retains $k = 12$ coefficients per head, while adaptive allocation distributes the same total coefficient budget across heads. Adaptive allocation improves Qasper from 22.23 to 24.07 and QMSum from 21.28 to 21.45. These results support allocating coefficients according to each token's KV vectors rather than assigning a fixed number to every head.

## 4 Related Work

Several related approaches compress KV caches through learned orthogonal projections or attention matching. We discuss how these approaches differ from OrthoKV in their representations and compression objectives.

**Orthogonal projections.** MatryoshkaKV (Lin et al., 2025) learns orthogonal projections through distillation and Matryoshka training to support adaptive KV cache compression. While both methods exploit orthogonality, MatryoshkaKV compresses KV vectors through low-dimensional projections, whereas OrthoKV uses complete orthogonal dictionaries to represent KV vectors with sparse coefficients.

**Attention matching.** Zweiger et al. (2026) propose an attention matching framework whose OMP variant selects a smaller set of keys and adjusts the values to preserve the original attention outputs. Although both approaches use sparse approximation, the role of sparsity differs: their method selects fewer keys from the cached sequence, whereas OrthoKV represents each KV vector with sparse coefficients over learned orthogonal dictionaries.

## 5 Limitations

Our evaluation is limited to two 7B–8B models with grouped-query attention (GQA). The effectiveness of OrthoKV on larger models and other attention architectures remains to be evaluated.

The benefits of OrthoKV are most evident under extreme KV cache compression, particularly for long contexts where the full-precision buffer accounts for a smaller fraction of cache storage. At compression levels comparable to those of 2-bit KIVI, OrthoKV generally achieves lower downstream quality. Thus, OrthoKV is particularly useful when memory constraints demand more aggressive compression than 2-bit KIVI provides.

## 6 Conclusion

We presented OrthoKV, a KV cache compression method that represents KV vectors with sparse coefficients over per-head complete orthogonal dictionaries. Orthogonality enables exact, non-iterative sparse coding and optimal coefficient allocation across heads. A sparsity-independent learning objective allows the same dictionaries to support multiple compression budgets. Experiments show that OrthoKV achieves comparable or better quality than Lexico in most evaluated settings, with smaller KV caches and substantially lower sparse coding latency. Compared with full KV caching, it also improves throughput at larger batch sizes and supports larger batches within the same GPU memory budget. These results demonstrate that overcomplete dictionaries are not necessary for effective sparse KV cache compression and that complete orthogonal dictionaries offer a practical alternative for extreme KV cache compression.

### AI USE STATEMENT

In this work, we used generative AI tools to assist with implementing the proposed method, debugging, and performance optimization. We also used generative AI tools to assist with literature search and discovery, and to improve the clarity, grammar, and readability of the manuscript. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text or artifacts produced with the aid of generative AI.

### REPRODUCIBILITY STATEMENT

Implementation details are provided in Section 3 and Appendix B. We also include demonstration code in the supplementary material.

## REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 3119–3137, 2024.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed Abdelfattah, and Kai-Chiang Wu. Palu: KV-cache compression with low-rank projection. In *International Conference on Learning Representations*, volume 2025, pp. 50222–50249, 2025.

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

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.

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. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, 2021.

Alexander Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1074–1084, 2019.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 113152–113188, 2025.

Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. Not all heads matter: A head-level KV cache compression method with integrated retrieval and reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 99269–99290, 2025.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 70–79, 2019.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. LongCoder: A long-range pre-trained language model for code completion. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 12098–12107, 2023.

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. In *Advances in Neural Information Processing Systems*, volume 37, pp. 1270–1303, 2024.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1601–1611, 2017.

Michel Journée, Yurii Nesterov, Peter Richtárik, and Rodolphe Sepulchre. Generalized power method for sparse principal component analysis. *Journal of Machine Learning Research*, 11 (15):517–553, 2010.

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

Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics*, 2002.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970, 2024.

Bokai Lin, Zihao Zeng, Zipeng Xiao, Siqi Kou, TianQi Hou, Xiaofeng Gao, Hao Zhang, and Zhijie Deng. MatryoshkaKV: Adaptive KV compression via trainable orthogonal projection. In *International Conference on Learning Representations*, volume 2025, pp. 86669–86690, 2025.

Tianyang Liu, Canwen Xu, and Julian McAuley. RepoBench: Benchmarking repository-level code auto-completion systems. In *International Conference on Learning Representations*, volume 2024, pp. 47832–47850, 2024a.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 32332–32344, 2024b.

S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993.

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

Y.C. Pati, R. Rezaiifar, and P.S. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pp. 40–44, 1993.

Yifei Shen, Ye Xue, Jun Zhang, Khaled Letaief, and Vincent Lau. Complete dictionary learning via $\ell_p$-norm maximization. In *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence*, volume 124, pp. 280–289, 2020.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads. In *International Conference on Learning Representations*, volume 2025, pp. 37228–37253, 2025.

Yuexiang Zhai, Zitong Yang, Zhenyu Liao, John Wright, and Yi Ma. Complete dictionary learning via l4-norm maximization over the orthogonal group. *Journal of Machine Learning Research*, 21 (165):1–68, 2020.

Hongxuan Zhang, Yao Zhao, Jiaqi Zheng, Chenyi Zhuang, Jinjie Gu, and Guihai Chen. CSR: achieving 1 bit key-value cache via sparse representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 25860–25867, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710, 2023.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. QMSum: A new benchmark for query-based multi-domain meeting summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5905–5921, 2021.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast KV compaction via attention matching. In *Proceedings of the Forty-third International Conference on Machine Learning*, 2026.

# A PROMPTS

## A.1 LONGBENCH TASK PROMPTS

We use the LongBench task prompts distributed with the KIVI (Liu et al., 2024b) evaluation code. The templates below show the task prompts before the model-specific chat template (where applicable). The placeholders {`context`} and {`input`} are filled with the corresponding dataset fields, including any demonstrations provided in the examples.

**Qasper**

You are given a scientific article and a question. Answer the question as concisely as you can, using a single phrase or sentence if possible. If the question cannot be answered based on the information in the article, write "unanswerable". If the question is a yes/no question, answer "yes", "no", or "unanswerable". Do not provide any explanation.

Article: {`context`}

Answer the question based on the above article as concisely as you can, using a single phrase or sentence if possible. If the question cannot be answered based on the information in the article, write "unanswerable". If the question is a yes/no question, answer "yes", "no", or "unanswerable". Do not provide any explanation.

Question: {`input`}

Answer:

**QMSum**

You are given a meeting transcript and a query containing a question or instruction. Answer the query in one or more sentences.

Transcript:
{`context`}

Now, answer the query based on the above meeting transcript in one or more sentences.

Query: {`input`}
Answer:

**MultiNews**

You are given several news passages. Write a one-page summary of all news.

News:
{`context`}

Now, write a one-page summary of all the news.

Summary:

**TREC**

Please determine the type of the question below. Here are some examples of questions.

{context}
{input}

**TriviaQA**

Answer the question based on the given passage. Only give me the answer and do not output any other words. The following are some examples.

{context}

{input}

**SAMSum**

Summarize the dialogue into a few short sentences. The following are some examples.

{context}

{input}

**LCC**

Please complete the code given below.
{context}Next line of code:

**RepoBench-P**

Please complete the code given below.
{context}{input}Next line of code:

### A.2 GSM8K PROMPT

GSM8K uses five training examples as demonstrations, followed by the test question. Each demonstration contains a worked solution and a final answer marked by `####`. The demonstrations and test question are combined into a single user message and formatted using the model-specific chat template.

**GSM8K**

Question: {demonstration_question_1}
Answer: {demonstration_solution_1}
#### {demonstration_final_answer_1}

*[Demonstrations 2–5 follow the same format.]*

Question: {test_question}
Answer:

## B IMPLEMENTATION DETAILS

**Sparse attention implementation.** We implement sparse QK computation with a Triton kernel. For AV, a custom CUDA kernel processes chunks of cached tokens and accumulates their contributions into partial coefficient buffers. A fused kernel then reduces these partial results and applies the value dictionary. Both computations use FP32 accumulation.

**Runtime optimizations.** During decoding, we use CUDA Graphs to reduce CPU kernel launch overhead in KV compression and attention computation. CUDA Graphs for KV compression are

reused across requests with compatible configurations. We combine key and value cache writes and reuse temporary buffers to reduce runtime overhead. We also keep an FP32 copy of each value dictionary to avoid repeated dtype conversion. We use eager attention during prefill for the downstream quality and evaluation-time experiments, and PyTorch SDPA during prefill for the batch-size scaling experiments.