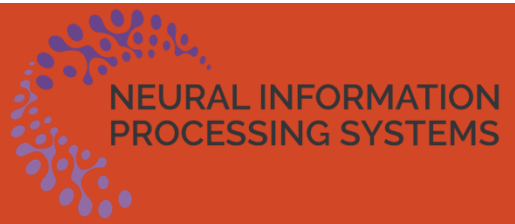


Value-Guided KV Compression for LLMs via Approximated CUR Decomposition

Ayan Sengupta, Siddhant Chaudhary, Tanmoy Chakraborty

KV Cache: What and Why?

- During generation, attention key and value matrices remain same for the context (input) tokens, only query changes. Recomputation of context key-value pair is redundant; might unnecessarily increase generation time.
- A KV cache can store the precomputed key-value matrices for the input tokens.
- Newly generated token key-value pairs can be dynamically added to the cache.

Why Compress KV Cache?

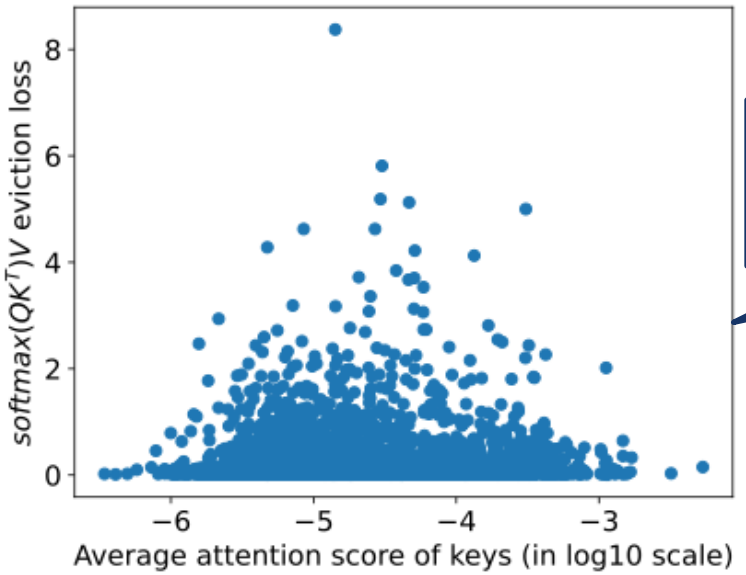
- For long-range generation, storing the entire KV cache for the entire duration of generation can lead to heavy GPU memory requirement (~several hundreds of GBs).
- Not all attention tokens are equally important for a generation step.
- Can we dynamically select which key-value pairs to be stored for each attention head?

KV Compression Methods for LLMs

- H2O, LLM Streamline, SnapKV, ChunkKV – several methods for compressing KV cache; reduce cache memory to <80GBs even for very long context (~128K) and long generation (~32K).
- Majority of these methods use heuristic on attention matrices (QK^T) – observed attention score for each attention key or expected attention score.
- Attention sink (first few tokens tend to have very high attention score with all queries) is also an effective heuristic to identify which key-value elements can be dropped from each attention head.
- Adaptive KV compression methods compress different attention heads from different layers with different compression ratios.

Why Value-aware KV Compression is Needed?

- Existing KV compression methods categorically ignore the attention value matrices.
- Due to SoftMax activation (which bounds the Lipschitz constant of the attention operation), attention values contain more differential importance on the final attention output.
- Average attention score of keys (a common heuristic used in existing methods) is not a good determinant of final attention importance.



No apparent correlation between attention score and eviction loss

Eviction loss is calculated as the $\|A - \hat{A}\|$, with A being original attention matrix and $\hat{A}$ attention matrix with evicted key and values

Value-aware KV Compression: *CurDKV*

- We propose value-aware KV compression method – *CurDKV*; which utilizes both key and value matrices to determine the unimportant indices for eviction/compression.
- CurDKV treats the compression problem as CUR decomposition; where $A \approx CUR$, with C being a column subset of A, U is a low-rank matrix and R is a row subset of A.
- A popular way to perform CUR decomposition is using leverage score (row-wise L2 norm of SVD matrices).
- SVD computation can be computationally expensive; another easier approximated method is to calculate row-wise L2 norm of GA, where G is a low-rank isotropic Gaussian matrix.
- CurDKV supports Group Query Attention, Flash attention; also supports adaptive budget allocation

Value-aware KV Compression: *CurDKV*

Algorithm 1: `CurDKV`: CUR-based KV Compression with GQA Support

Input: Key matrix $K \in \mathbb{R}^{g \times n \times d}$, Value matrix $V \in \mathbb{R}^{g \times n \times d}$, group budget k, Gaussian projection dim r, num_sink s

Output: Compressed keys $K' \in \mathbb{R}^{g \times k \times d}$, values $V' \in \mathbb{R}^{g \times k \times d}$

for *each group* g_i *in* $1, \ldots, g$ **do**

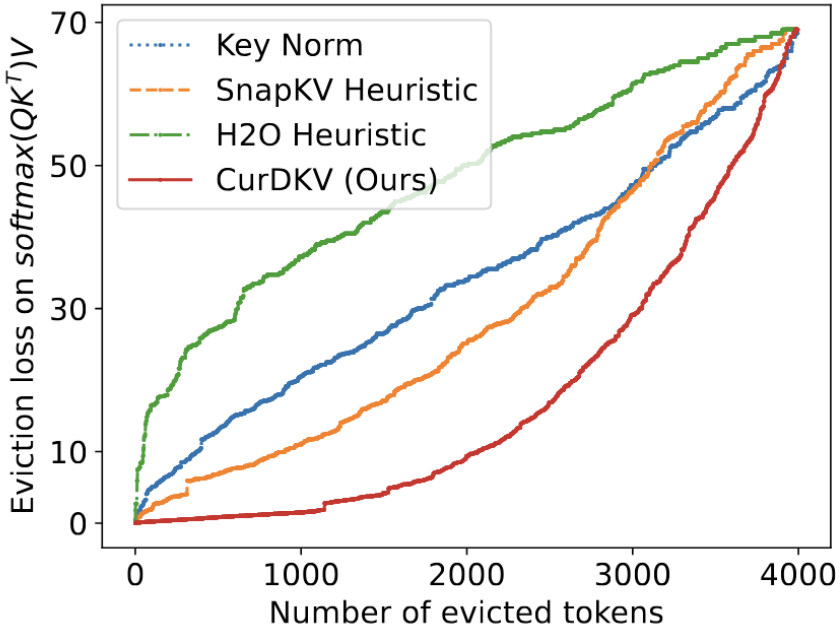
- Sample Gaussian matrix $G \in \mathbb{R}^{d \times r}$ with $G_{ij} \sim \mathcal{N}(0, 1/r)$;
- Project $K_i \leftarrow K_i G$, $V_i \leftarrow V_i G$;
- Compute key leverage scores: $\ell_j^{(K)} = \|K_i[j]\|_2^2$, value leverage: $\ell_j^{(V)} = \|V_i[j]\|_2^2$;
- Combine scores: $\ell_j^{(KV)} = \ell_j^{(K)} \cdot \ell_j^{(V)}$;
- Normalize: $\tilde{\ell}_j = \ell_j^{(KV)} / \sum_j \ell_j^{(KV)}$;
- Preserve first s tokens as sink indices: $S_{\text{sink}} = \{0, \ldots, s-1\}$;
- Select top-$(k-s)$ indices from $\tilde{\ell}[s:]$: $S_{\text{top}} = \text{TopK}(\tilde{\ell}[s:], k-s) + s$;
- Combine: $S_i = S_{\text{sink}} \cup S_{\text{top}}$;
- $K_i' \leftarrow K_i[S_i]$, $V_i' \leftarrow V_i[S_i]$;

return K', V'

Value-aware KV Compression and Eviction Loss

Lemma 3.1. *Let $Q, K, V \in \mathbb{R}^{n \times d}$ be the query, key and value matrices in the attention computation. Further, let $K', V' \in \mathbb{R}^{n \times d}$ be sub-matrices obtained by zeroing-out a subset of rows of K and V respectively. Then,* [2]

$$\|softmax(QK^T)V - softmax(QK'^T)V'\|_F \leq \sqrt{n}\|V - V'\|_F + 2\sqrt{n}\|V'\|_F \quad (3)$$



Above lemma and the visualization shows the value-aware method can achieve lower eviction loss

Empirical Evidence

	Task	Chunk	KNorm	Snap	Streaming	CurDKV
30%	2WikiMQA	51.64	40.14	53.85	34.10	**58.07**
	HotpotQA	58.64	46.94	57.52	47.15	**62.60**
	MF-en	45.41	43.39	**48.56**	32.61	50.42
	Qasper	42.46	37.73	41.86	43.11	**47.75**
	Average	49.54	42.05	50.45	39.24	**54.71**
90%	2WikiMQA	24.24	6.14	9.84	18.54	**40.59**
	HotpotQA	37.85	3.80	36.29	30.38	**43.51**
	MF-en	24.31	13.12	23.04	22.03	**31.24**
	Qasper	13.62	7.02	14.29	14.56	**16.10**
	Average	25.00	7.52	24.10	21.38	**32.86**

(a) LongBench (Qwen2.5-32B, 4-bit)

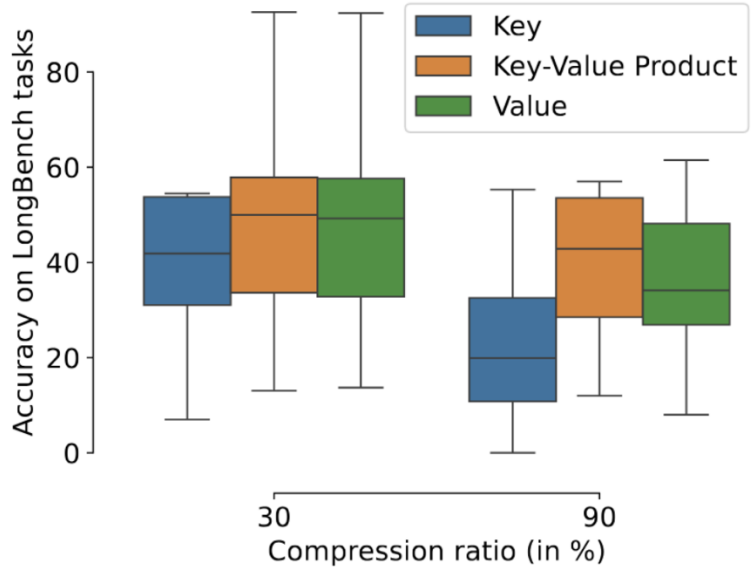
On LongBench tasks CurDKV achieves best performance retention for Qwen-32B model. Similar results observed with Qwen-14B, LLaMA-8B and Mistral-7B.

Even for long context (up to 128K) needle-in-a-haystack tasks CurDKV and Adaptive CurDKV achieves competitive performance

	Task	ChunkKV	SnapKV	CurDKV	AdaSnapKV	AdaCurDKV
		Non-Adaptive Methods			Adaptive Methods	
30%	S-1	**100.0** / 93.6	**100.0** / 54.8	**100.0** / 74.2	**100.0** / 64.5	**100.0 / 85.5**
	S-2	**100.0** / 91.2	**100.0** / 63.2	**100.0** / 91.2	**100.0 / 80.7**	**100.0 / 96.5**
	S-3	**96.1** / 56.9	25.5 / 2.0	94.1 / 72.6	82.4 / 13.7	**88.2 / 60.8**
	MK-1	**100.0** / 89.6	**100.0** / 41.7	**100.0** / 91.7	**100.0 / 77.1**	**100.0 / 93.8**
	MK-2	57.8 / 57.8	71.1 / 44.4	**100.0 / 93.3**	97.8 / **84.4**	**100.0 / 84.4**
	MK-3	44.7 / **46.8**	53.2 / 29.8	**95.7** / 23.4	89.4 / **68.1**	**93.6** / 44.7
	MQ	**100.0 / 85.1**	99.5 / 31.7	99.5 / 82.2	**100.0** / 58.7	**100.0 / 83.7**
	MV	97.4 / 88.8	94.1 / 31.6	**100.0 / 93.4**	**100.0** / 64.5	**100.0 / 95.4**
	Average	87.0 / 76.2	80.4 / 37.4	**98.7 / 77.8**	96.2 / 64.0	**97.7 / 80.6**

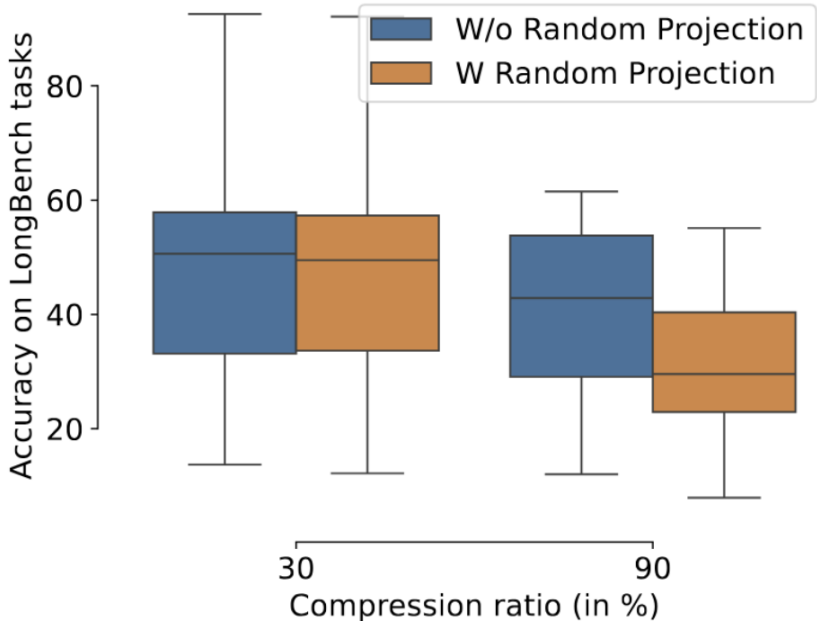
(b) Needle-in-a-haystack (Llama-3.1-8B & Mistral-7B)

Ablation with CurDKV



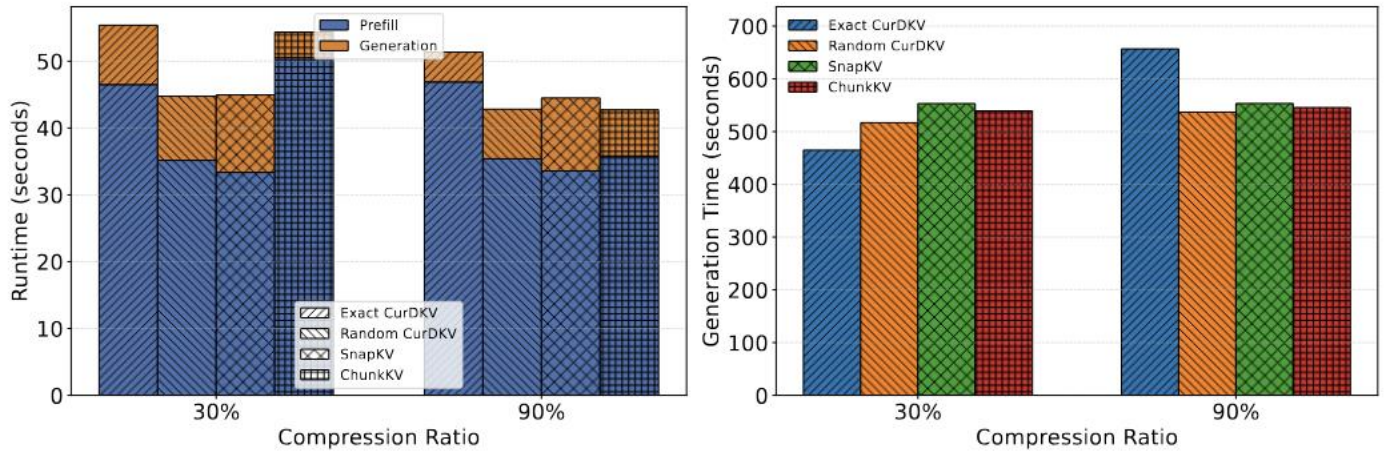
Only key matrices to determine KV importance leads to suboptimal results; particularly at higher compression rate

Albeit computationally cheaper, random leverage computation can lead to suboptimal performance



Computational Benefit of CurDKV

Method	Total	Prefill	Generation
Exact CurDKV	55.39	46.51	8.88
Random CurDKV	**44.76**	35.17	9.59
SnapKV	44.97	**33.37**	11.59
ChunkKV	54.36	50.53	**3.83**



Exact or random CurDKV typically requires higher generation time than methods like ChunkKV, due to computation of leverage scores.

However, prefilling time remain moderately low for CurDKV.

Random CurDKV is faster than exact CurDKV during prefilling; not so effective during generation

Key takeaways

- Use exact CurDKV for higher compression ratio and long generation tasks
- Use random CurDKV for lower compression ratio/shorter-range generation tasks

Next Steps

- Explore **learned or hybrid CUR decompositions** using neural predictors for leverage scores instead of random projections. Get the best out of exact and random CurDKV.
- Extend CurDKV to **multimodal and multilingual LLMs** to test robustness across modalities and languages.
- Formulate **scaling laws for KV compression** linking leverage-score variance, compression ratio, and performance degradation.