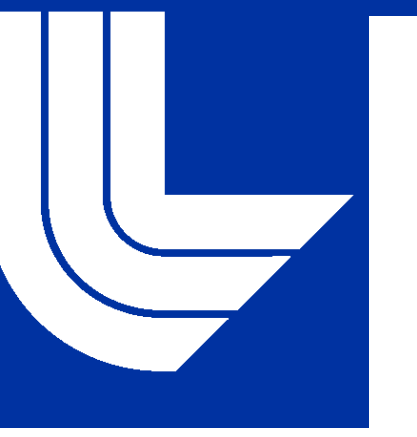




# Query-to-Query Attention: Bandwidth-Efficient LLM Inference via Sparse Attention



Aditya Tomar, Brian Bartoldson, Bhavya Kailkhura

The KV cache is the primary bottleneck for LLM inference, especially for long context lengths and large batch sizes. To address this, we develop a novel method that utilizes the relationship between queries to determine which keys and values to load from the KV cache. This allows us to perform sparse attention, reducing the number of memory operations and speeding up LLM inference.

## INTRODUCTION

During LLM inference, generating each new token requires loading and storing the entire KV cache to and from GPU memory. The KV cache grows linearly with sequence length and batch size. For long context lengths and large batch sizes, the KV cache uses more memory than the model weights, ultimately dominating latency as it becomes very expensive to load and store.

$$\text{Attention} = \text{softmax}\left(\frac{\vec{q}K^T}{\sqrt{h_d}}\right) \cdot V$$

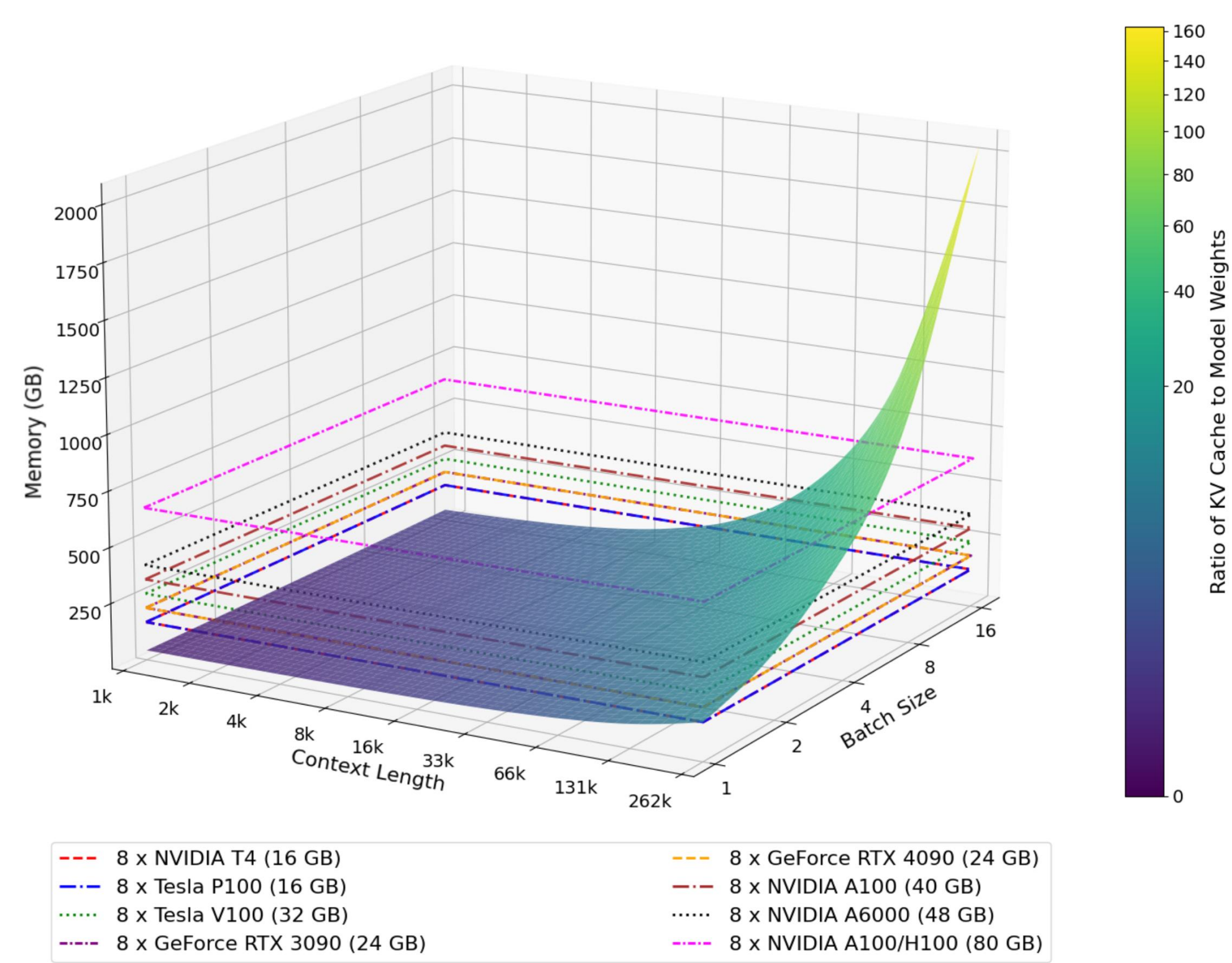


Figure 1. KV cache memory usage by Llama-2-7B on a single node as context length and batch size are scaled logarithmically. [1]

## IDEA

Can we use the queries to determine which keys and values to load?

### Observations

- Queries are low rank (keys are also low rank) [3]
- Queries have very high cosine similarity with each other

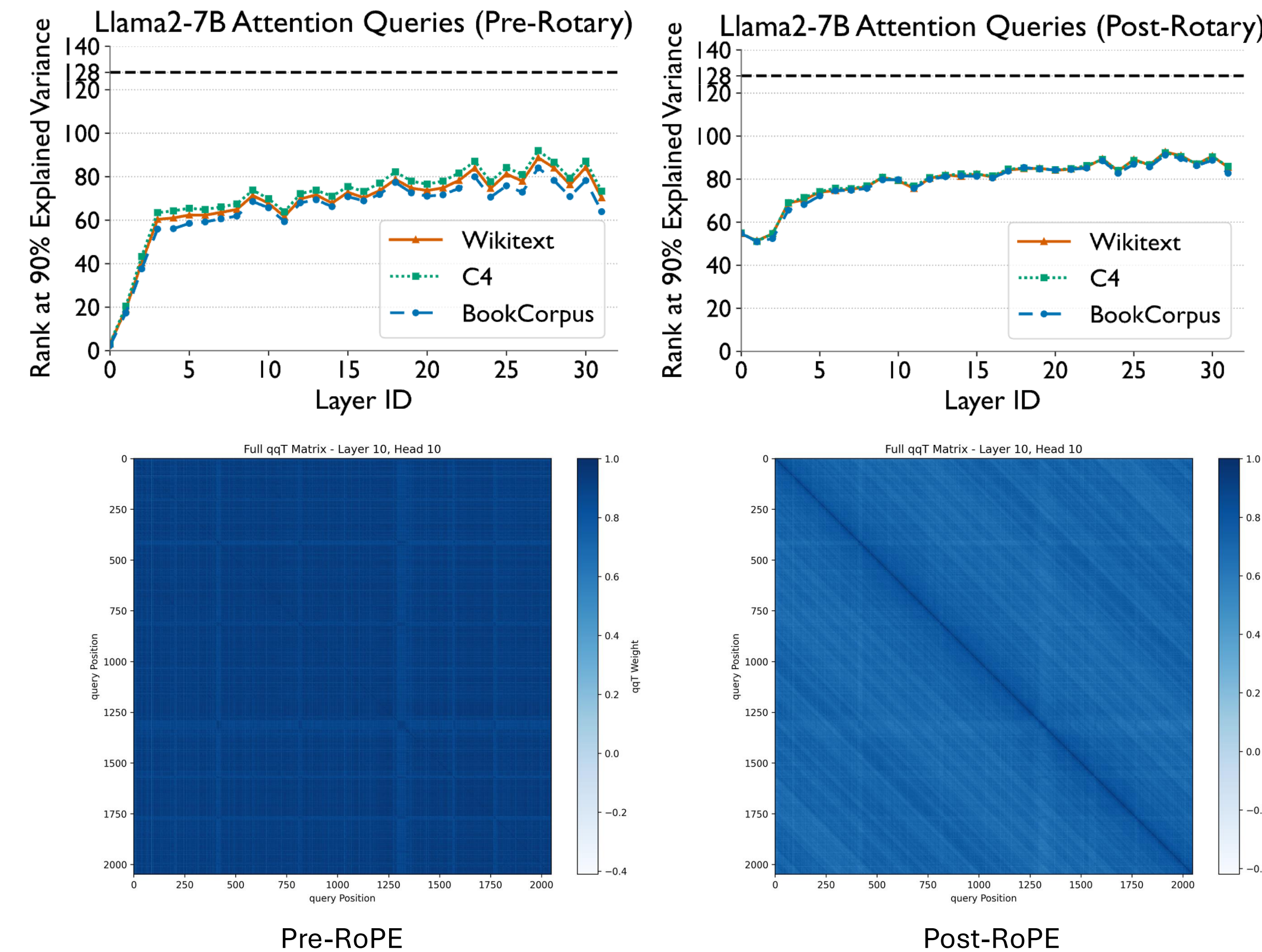


Figure 2. (Top) Rank of queries for Llama-2-7B model on different datasets [3]. (Bottom) Pairwise cosine similarities of queries for Llama-3.1-8B on a 2K sequence length sample from WikiText2 dataset.

## RESULTS

### Memory Operations

Vanilla Attention:

$$2 \cdot L \cdot h_d$$

Query-to-Query Attention:

$$c \cdot h_d + c \cdot \text{top}_k + 2 \cdot \text{top}_q \cdot \text{top}_k \cdot h_d = c \cdot (h_d + \text{top}_k) + 2 \cdot \text{top}_q \cdot \text{top}_k \cdot h_d$$

### Experiments

Evaluated our method using perplexity for different models on WikiText-2 dataset. The last 128 tokens of pre-fill are computed using this approximate method. The cached queries are the 128 queries preceding the last 128 approximated tokens. The number of top-q chosen queries is 5, and the number of top-k chosen keys is 128. The head dimension is 128 for each of the models. So  $c = h_d = \text{top}_k = 128$  and  $\text{top}_q = 5$ .

| Context Length | Method          | Llama-3.1-8B |      | Mistral-7B |      | Llama-2-7B-32K |      | Memory Ops. Reduction |
|----------------|-----------------|--------------|------|------------|------|----------------|------|-----------------------|
|                |                 | Wiki         | C4   | Wiki       | C4   | Wiki           | C4   |                       |
| 2048           | Baseline        | 6.24         | 9.54 | 5.32       | 8.47 | 5.76           | 7.72 | 1×                    |
|                | Query Attention | 6.28         | 9.61 | 5.33       | 8.50 | 5.88           | 7.91 | 2.7×                  |
| 4096           | Baseline        | 5.84         | 9.28 | 4.95       | 8.04 | 5.38           | 7.46 | 1×                    |
|                | Query Attention | 5.86         | 9.33 | 4.96       | 8.05 | 5.44           | 7.59 | 5.3×                  |
| 8192           | Baseline        | 5.61         | 9.10 | 4.75       | 7.86 | 5.16           | 7.31 | 1×                    |
|                | Query Attention | 5.62         | 9.11 | 4.75       | 7.87 | 5.19           | 7.38 | 10.7×                 |

## NEXT STEPS

- Implement this method for generation to be able to run downstream task evaluations
- Implement shifting dictionary of cached queries
- Experiment with picking representative queries for earlier parts of the context
- Compare against other sparse attention methods

## CONTACT

Aditya Tomar | (408) 425-6656 | adityatomar@berkeley.edu

### REFERENCES

[1] Tiwari, Rishabh, et al. "QuantSpec: Self-Speculative Decoding with Hierarchical Quantized KV Cache." *arXiv preprint arXiv:2502.10424* (2025). [2] Zhang, Zhenyu, et al. "H2o: Heavy-hitter oracle for efficient generative inference of large language models." *Advances in Neural Information Processing Systems* 36 (2023): 34661-34710. [3] Singhania, Prajwal, et al. "Loki: Low-rank keys for efficient sparse attention." *Advances in Neural Information Processing Systems* 37 (2024): 16692-16723. [4] Ribar, Luka, et al. "Sparq attention: Bandwidth-efficient llm inference." *arXiv preprint arXiv:2312.04985* (2023).

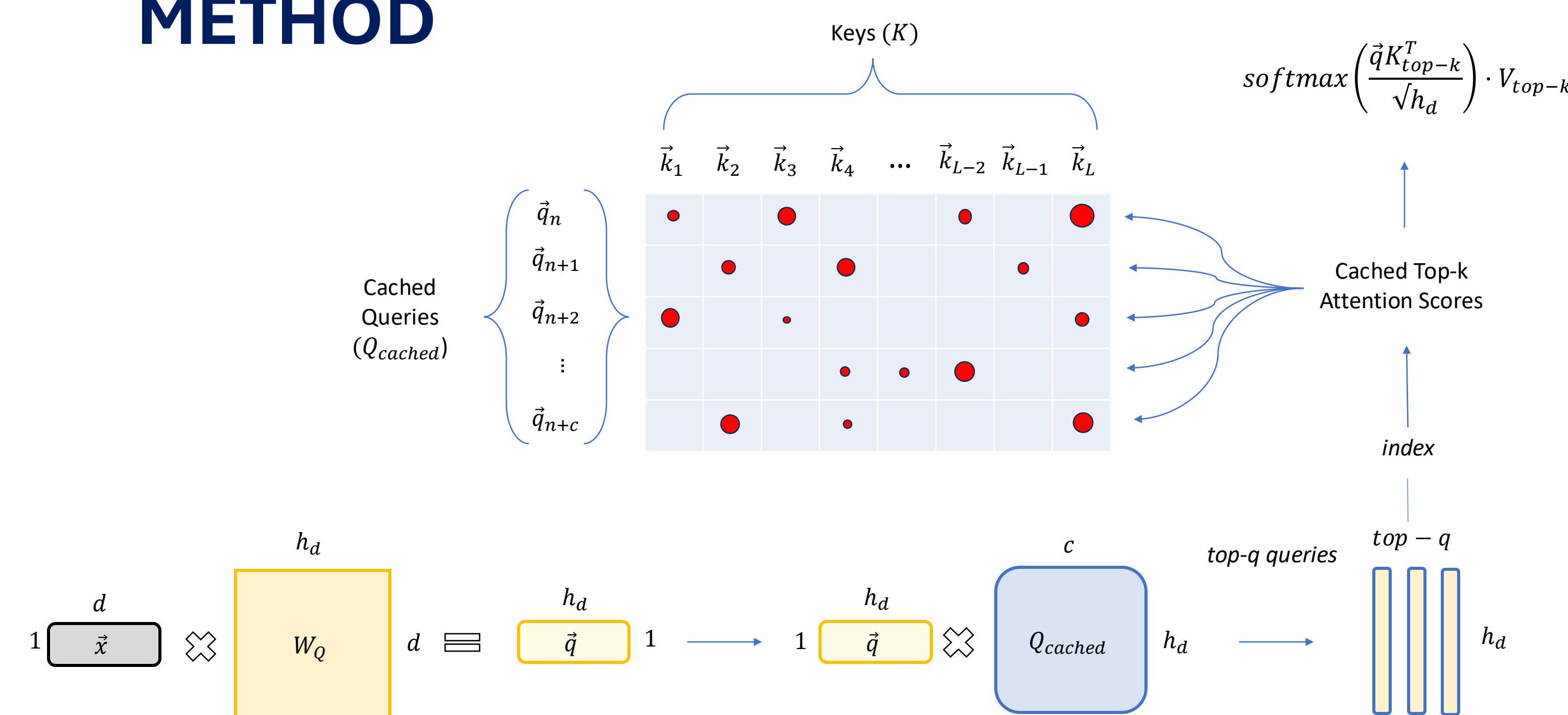
## Motivating Question

Can we selectively load only the important keys and values at each inference step to reduce the number of memory operations?

## RELATED WORKS

- **H<sub>2</sub>O**: Maintains a running sum of attention scores and only loads keys and values for highest cumulative attention scores [2]
- **Loki**: Projects queries and keys into a lower-dimensional space using top-p principal directions via PCA, performs attention in this subspace to identify top-k important tokens, and then performs full attention only for those tokens [3]
- **SparQ**: Identifies the largest elements of each query vector, and only loads the associated elements from each of the key vectors to compute attention [4]

## METHOD



- 1) Cache a subset of queries and the indices for their top-k attention scores
- 2) For each new query, compute the cosine similarity between it and cached queries
- 3) Pick the top-p queries, then only load the top-k keys and values for those queries