Quest: Query-Aware Sparsity for Efficient Long-Context LLM Inference

Jiaming Tang [*1 2] **Yilong Zhao** [*1 3] **Kan Zhu** [3] **Guangxuan Xiao** [2] **Baris Kasikci** [3] **Song Han** [2 4]

Abstract

As the demand for long-context large language models (LLMs) increases, models with context windows of up to 128K or 1M tokens are becoming increasingly prevalent. However, long-context LLM inference is challenging since the inference speed decreases significantly as the sequence length grows. This slowdown is primarily caused by loading a large KV cache during self-attention. Previous works have shown that a small portion of critical tokens will dominate the attention outcomes. However, we observe the criticality of a token highly depends on the query. To this end, we propose Quest, a query-aware KV cache selection algorithm. Quest keeps track of the minimal and maximal Key values in KV cache pages and estimates the criticality of a given page using Query vectors. By only loading the Top-K critical KV cache pages for attention, Quest significantly speeds up self-attention without sacrificing accuracy. We show that Quest can achieve up to 7.03× self-attention speedup, which reduces inference latency by 2.23× while performing well on tasks with long dependencies with negligible accuracy loss. Code is available at https://github.com/mit-han-lab/Quest.

1. Introduction

The rapid evolution of Large Language Models (LLMs) has shaped our daily lives. With the increasing demand for multi-round conversations and long document queries, the maximum context length of LLMs has dramatically grown from 2K to 1M (Liu et al., 2024a; Peng et al., 2023; Tworkowski et al., 2023). The 128k context length GPT-4 model has already been deployed in large-scale serving, which is equivalent to 300 pages of text (OpenAI, 2023).

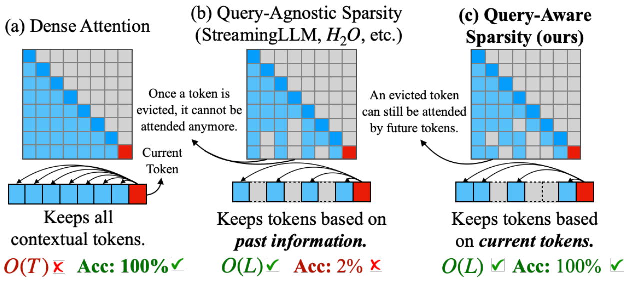

(a) Dense Attention
(b) Query-Agnostic Sparsity (StreamingLLM, H_2O, etc.)
(c) Query-Aware Sparsity (ours)
Once a token is evicted, it cannot be attended anymore.
Current Token
An evicted token can still be attended by future tokens.
Keeps all contextual tokens.
$O(T)$ ✗ Acc: 100% ✓
Keeps tokens based on *past information.*
$O(L)$ ✓ Acc: 2% ✗
Keeps tokens based on *current tokens.*
$O(L)$ ✓ Acc: 100% ✓

Figure 1. Comparison between Dense Attention(a), Query-Agnostic Sparsity (b) and Quest's Query-aware Sparsity (c). Quest significantly speeds up self-attention while maintaining high accuracy by dynamically determining the critical tokens based on the current query. T represents the total sequence length and L represents the number of critical tokens for attention.

However, processing long-context requests is challenging. Due to the auto-regressive nature of LLMs, generating one token would require reading the entire KV cache. For Llama 7B model (Touvron et al., 2023) with 32k context length, the KV cache can occupy 16GB of space, which requires at least 11 ms to read, which contributes to more than 50% of the inference latency[*], limiting the overall throughput.

Despite the increasingly large size of the KV cache, previous works have shown that a small portion of the tokens can dominate the accuracy of token generation (Zhang et al., 2023b; Ge et al., 2024). Therefore, we can dramatically reduce the inference latency by only loading the critical tokens, while still maintaining accuracy. Thus, it is essential to identify critical portions of the KV cache.

In this work, we further observe that the criticality of the tokens can change with different query tokens. As shown in Fig. 2, the critical tokens vary a lot with different queries. Therefore, we need a dynamic and efficient approach to determine which portion of the KV cache needs to be attended to. To this end, we propose Quest, a query-aware criticality estimation algorithm for long-context LLM inference that efficiently and effectively identifies critical KV cache tokens and performs self-attention selectively on chosen tokens, as shown in Fig. 1.

To reduce the overhead of KV cache criticality estimation,

[*]Equal contribution [1]Shanghai Jiao Tong University [2]MIT [3]University of Washington [4]NVIDIA. Correspondence to: Song Han <songhan@mit.edu>, Baris Kasikci <baris@cs.washington.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[*]Tested with FP16 FlashInfer implementation on an RTX4090

Quest manages KV cache at page granularity (Kwon et al., 2023). For each page, Quest utilizes maximum and minimum values of each feature dimension of the Key vector as the metadata to represent token information. During inference, Quest considers both the Query vector and the metadata to estimate each page's criticality. Given all criticality scores of the pages, Quest chooses Top-K pages to perform approximate self-attention, where K is a preset constant (e.g. 128, 256). By reducing the memory movement from the entire KV cache to metadata and constant K pages, Quest significantly accelerates inference.

We evaluate both the accuracy and efficiency of Quest. Since Quest dynamically decides the criticality of the tokens, Quest achieves better accuracy for a given degree of KV cache sparsity than baselines on PG19 dataset (Rae et al., 2019), passkey retrieval task (Peng et al., 2023), and LongBench (Bai et al., 2023) with 256 to 4K token budgets. For 32K context, Quest achieves 7.03× self-attention latency reduction compared to FlashInfer (Ye et al., 2024). Our end-to-end framework demonstrates that Quest can have 2.23× inference speedup compared to FlashInfer (Ye et al., 2024) with 4-bit weight quantization. In summary, we make the following contribution:

- An analysis of the self-attention mechanism that pinpoints the importance of query-aware sparsity.
- Quest, an efficient and accurate KV cache acceleration algorithm, which exploits query-aware sparsity by dedicated operator designs and implementations.
- A comprehensive evaluation of Quest, demonstrating up to 7.03× self-attention latency reduction and 2.23× end-to-end latency improvement.

2. Related Work

2.1. Long-context Model

As the demand for long-context models increases, many works have focused on extending the context window of LLMs. Currently, many models utilize Rotary Position Embeddings (RoPE) (Su et al., 2023), and by different scaling methods of RoPE with fine-tuning, the window size of the original 4k Llama-2 has been expanded to 32k for LongChat (Li et al., 2023) and 128k for Yarn-Llama-2 (Peng et al., 2023). Through length extrapolation, the context windows of models reached beyond 1M (Liu et al., 2024b). Beyond open-source models, GPT-4 Turbo supports lengths of up to 128k, while Claude-2 supports up to 200k (OpenAI, 2024; Anthropic, 2024). With models increasingly capable of handling long input, this poses challenges for inference efficiency. Quest aims to boost long-context inference by exploiting query-aware KV cache sparsity.

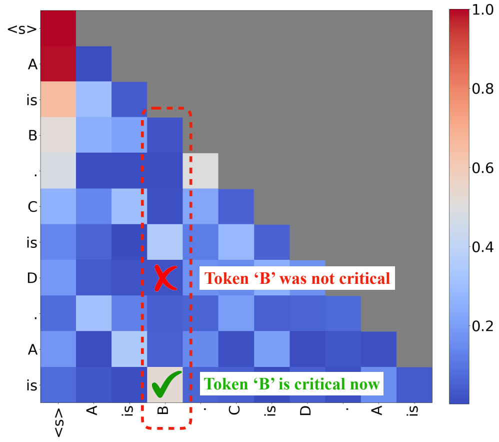


Figure 2. The attention map of prompt "A is B. C is D. A is". Each row represents the attention scores of previous tokens queried by the tokens on the left. When queried with "D", token "B" has a low attention score, showing "B" is not critical for generation. However, the "is" strongly attends to "B". Therefore, the criticality of tokens strongly correlates with the current query token.

2.2. KV Cache Eviction Algorithm

For long-context LLM inference and serving scenarios, the huge size of the KV cache results in significant time and space overheads. Many previous efforts have been dedicated to compressing the size of the KV cache to accelerate attention and reduce memory usage. H2O (Zhang et al., 2023b) retains a limited budget of the important KV cache based on the sum of historical attention scores. FastGen (Ge et al., 2024) further refines the types of tokens, applying a more sophisticated strategy for selecting the KV cache to keep. TOVA (Oren et al., 2024) simplifies the policy by deciding which tokens to permanently discard based solely on the current query. StreamingLLM (Xiao et al., 2023) handles infinitely long texts with attention sinks and a finite KV cache. These methods decide which parts of the KV cache to discard based on historical information or current states, but discarded tokens might be important for future tokens, which may cause the loss of important information. To mitigate this issue, SparQ (Ribar et al., 2023) computes approQximate attention scores by channel pruning and selects important tokens through them. However, this approach has not been widely validated for tasks with long dependencies, and the channel-level sparsity might pose challenges to practical acceleration. Therefore, we propose Quest, which retains all of the KV cache and selects part of the KV cache based on the current query to accelerate long-context self-attention without accuracy degradation.

3. Methodlogy

In this section, we first motivate Quest by analyzing the breakdown of inference cost and self-attention properties.

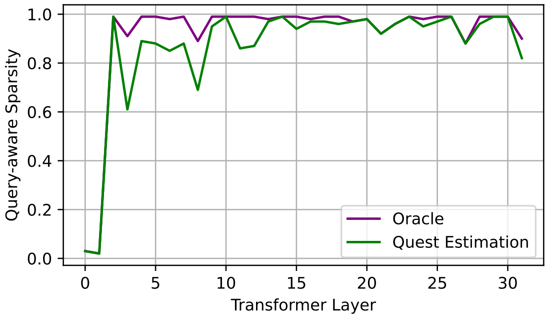


Figure 3. The query aware sparsity for each layer in LongChat-7B model. We measure the sparsity by eliminating KV cache tokens while making sure the perplexity on PG19 increases less than 0.01. For the first two layers, the sparsity is below 10%, while for the rest of the layers, the sparsity is larger than 90%, showing great potential for optimization. Quest closely aligns with the oracle.

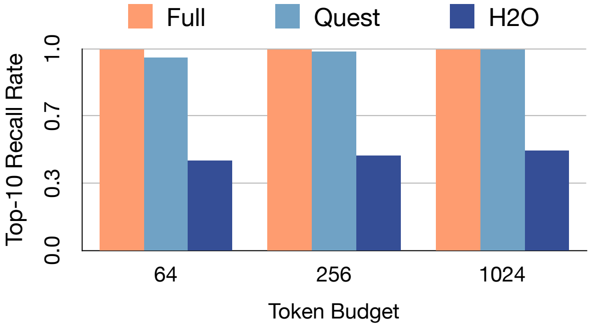


Figure 4. Recall rate of tokens with Top-10 attention scores. Results are profiled with LongChat-7b-v1.5-32k model in passkey retrieval test of 10K context length. Recall rate is the ratio of tokens selected by different attention methods to tokens selected by the full attention in each round of decoding. The average rate is shown in the figure, with various token budgets assigned.

We then present the design of Quest and discuss its benefits.

3.1. Long-context Inference Is Costly

LLM inference contains two stages, namely, the prefill stage and the decode stage. In the prefill stage, all the input tokens are transformed into embeddings and generate the Key (K), Query(Q), and Value(V) vectors. Both the Key and the Value vectors are saved in the KV cache for future use. The rest of the prefill stage includes self-attention and feed-forward network (FFN) layers, which produce the first response token.

In the decode stage, the model will take the last generated token to calculate its K, Q, V. The model uses Q to multiply with every K of previous tokens to generate the *attention weights*. The attention weights will then get normalized using softmax, where each value a_i represents the attention score between ith token and the current token. The self-attention layer will output $\sum a_i \cdot V_i$ and send to the FFN.

For one request, the prefill stage only happens once, while a decoding process is needed for every token in the response. Therefore, the decode stage dominates the inference time. For example, for 16k token prompts and 512 token responses, over 86% of the time is spent on decode stages. Therefore, the decode stage performance is crucial for overall latency.

Moreover, a long-context scenario significantly slows down the decode stage. In every decode stage, the K and V of existing tokens must be loaded to perform self-attention, which can easily reach 16GB for the 32k context of Llama-7b[†]. This memory load operation can take 53% of the time in a decode stage. Therefore, optimizing self-attention becomes a must for efficient long-context inference.

3.2. Self-Attention Operation Features High Sparsity

Luckily, previous research has highlighted the inherent sparsity in self-attention (Zhang et al., 2023b; Ge et al., 2024). Due to this property of self-attention, a small portion of tokens in the KV cache, called critical tokens, can accumulate sufficient attention scores, capturing the most important inter-token relationships. For example, as shown in Fig. 3, apart from the first two layers, less than 10% of the tokens are needed to achieve similar accuracy, which makes the attention on the rest of the tokens unnecessary. Therefore, if we can estimate the criticality of the tokens, we can only compute self-attention on critical KV cache tokens to greatly reduce the memory movement and thus improve efficiency.

3.3. Critical Tokens Depend on the Query

However, the criticality of the tokens is dynamic and highly dependent on the query vector Q. Assuming the prompt is ”A is B. C is D. A is”, we demonstrate the attention map of a certain head in the 16th layer of Llama-2-7b in Fig. 2. Since the output answer here should be ”B”, the token ”B” is critical to the current query ”is”. Thus, it has a high attention score. However, before the final token ”is”, ”B” is not critical for any previous query and has very low attention scores. In other words, the criticality of tokens is tightly related to the query token.

We quantify this effect by profiling the average recall rate of tokens with Top-10 attention scores along the text generations. The original attention with full KV cache can maintain 100% recall rate. However, KV cache eviction algorithm like H2O (Zhang et al., 2023b) which prunes tokens based on history information, suffers from low recall

[†] KV cache size $=$ 2 (both K and V) $*$ Num of Layer $*$ Sequence length $*$ Num of Heads $*$ Head Dimensions $*$ Size of FP16 $= 2 * 32 * 32 * 32 * 128 * 2 = 16$GB

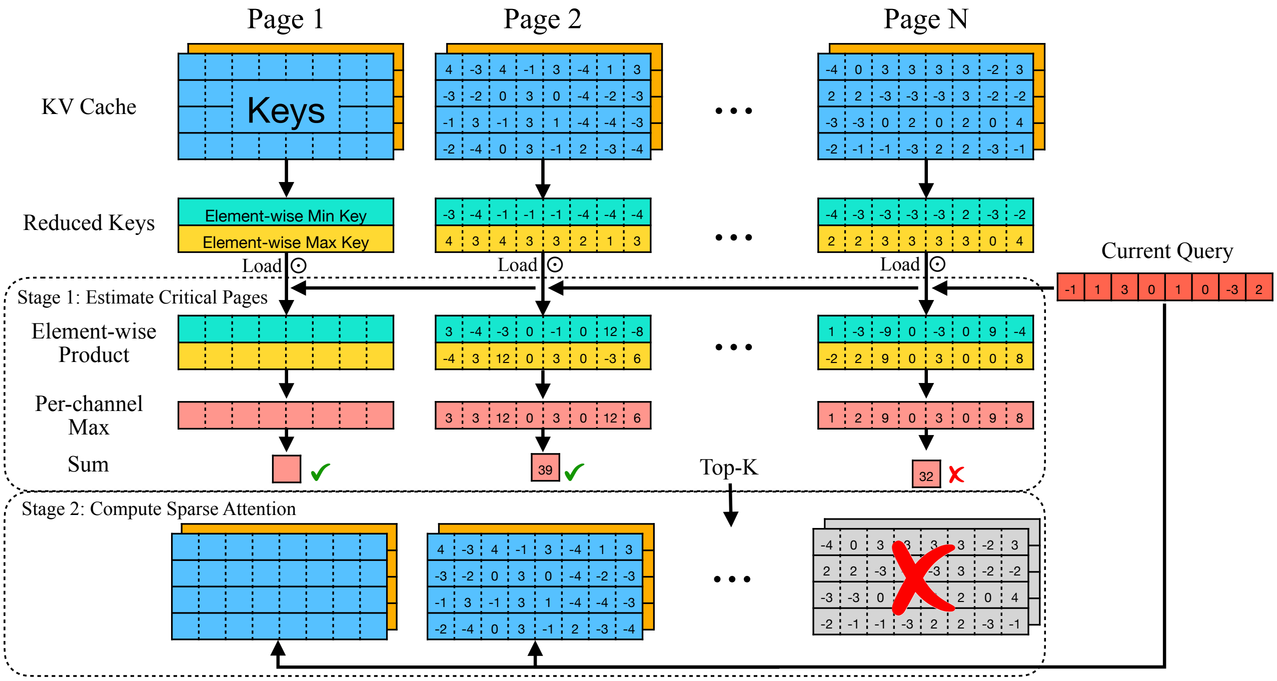


Figure 5. Quest performs self-attention in two stages. In stage 1, Quest estimates the criticality of pages by performing element-wise product between the current Query vector and both Min Key and Max Key vectors in each KV cache page. Quest gets the sum of the per-channel maximal value for each page as the page criticality estimation. In stage 2, only Top-K KV cache pages are loaded to perform sparse self-attention with the current Query.

rates since critical tokens are pruned in previous iterations. As shown in Fig. 4, Quest maintains recall rate close to full attention, as it estimated critical tokens based on current query. Therefore, pre-determining the criticality is challenging, which motivates query-aware sparsity by considering Q vectors for criticality estimation.

3.4. Dynamically Estimating Token Criticality

To efficiently and accurately estimate the criticality of KV cache tokens, we propose Quest, an efficient and accurate algorithm that exploits query-aware context sparsity, which approximately selects the most potentially critical KV cache pages for the current query. We show the workflow of Quest in Fig. 5. To manage the overhead, Quest adopts PageAttention (Kwon et al., 2023) and selects the KV cache pages at the granularity of pages.

To estimate the criticality of the pages, Quest performs an approximate calculation of attention weights before the original attention operation, as shown in Algorithm 1.

Our insight is that in order not to miss critical tokens, we should select pages containing the token with the highest attention weights. However, for an efficient selection of pages, we should calculate an approximate attention score following this insight. We found that the upper bound attention weights within a page can be used to approximate the highest attention in the page. The upper bound of the attention weights can be calculated by the channel-wise minimal values (m_i) and maximal values (M_i) of Key vectors. Given a Q vector, Quest calculates the maximum possible value of the channel i by taking $U_i = \max(Q_i m_i, Q_i M_i)$. Note that U_i is always greater than any product of Q_i with the Key value K_i for all tokens in this page regardless of the sign of Q_i. Therefore, when we add up U_i, we get the upper bound of attention weights across all Key vectors on this page.

After deriving the upper bound attention weights, we choose the top K pages as critical, where K is an arbitrarily defined hyper-parameter. To demonstrate the feasibility of Quest, we perform actual self-attention and gather Top-K per-page attention scores. As shown in Fig. 3, our query-aware sparsity mostly aligns with the oracle sparsity. Quest performs normal self-attention only on selected pages, which greatly reduces memory movement. We define the number of tokens in selected pages as the "Token Budget".

Due to the low sparsity ratio for the first two layers (as shown in Fig. 3), we only apply Quest and all baselines on later layers to better preserve model accuracy. Note that whether to skip the first two layers or not is orthogonal to the KV cache selection algorithm.

Algorithm 1 Token Criticality Estimation

```
When inserting new token to KV cache:
Input: Key vector $K$, Dimension of hidden states $dim$,
Current maximal vector $M_i$, Current minimal vector $m_i$

for $i = 1$ to $dim$ do
    $M_i = \max(M_i, k_i)$
    $m_i = \min(m_i, k_i)$
end for

When perform self-attention:
Input: Query vector $Q$, Dimension of hidden states $dim$,
Current maximal vector $M_i$, Current minimal vector $m_i$

Initialize $score = 0$.
for $i = 1$ to $dim$ do
    $score$ += $MAX(q_i * max, q_i * min)$
end for
```

3.5. Quest Reduces the Memory Movement of Self-Attention

Instead of loading the whole KV cache, Quest only needs to load a fraction of the data, which leverages query-aware sparsity. Assume that every K or V vector is M bytes, the KV cache contains L tokens, and each page contains S KV pairs (Page size). During criticality estimation, Quest will load maximal and minimal vectors of each page, which is approximately $2M * L/S$ bytes. Additionally, Quest performs normal self-attention for top K pages, which is $2M * K * S$ bytes. The whole KV cache is $2M * L$ bytes, which indicates Quest loads $1/S + K * S/L$ of the total KV cache[‡], which is equivalent to

$$\frac{1}{\text{Page Size}} + \frac{K}{\text{Page Num}}$$

Assuming that we use 16 KV pairs per page, context length is 64K, and we choose the top 4K pages, Quest will reduce the memory load by 8×. Note that this memory load reduction is universal across all models and is compatible with existing quantization mechanisms (Zhao et al., 2024).

4. Experiments

4.1. Setting

We evaluate Quest on the language modeling dataset PG19 (Rae et al., 2019), passkey retrieval task (Peng et al., 2023), and six datasets in LongBench (Bai et al., 2023): NarrativeQA (Kočiský et al., 2018), HotpotQA (Yang et al., 2018), Qasper (Dasigi et al., 2021), TrivialQA (Joshi et al., 2017), GovReport (Huang et al., 2021), MultifieldQA (Bai et al., 2023). We choose two widely used long-context models for our evaluation: LongChat-v1.5-7b-32k (Li et al., 2023) and Yarn-Llama-2-7b-128k (Peng et al., 2023). We compare our method against the KV cache eviction algorithm H2O (Zhang et al., 2023b), TOVA (Oren et al., 2024), and StreamingLLM (Xiao et al., 2023). Note that we **do not** apply any Quest and other baseline algorithms to the first two layers of the model, as our analysis in Sec 3.4 indicates a low sparsity ratio for these layers.

Method / Budget	32	64	128	256	512
H2O	0%	1%	1%	1%	3%
TOVA	0%	1%	1%	3%	8%
StreamingLLM	1%	1%	1%	3%	5%
Quest (ours)	**65%**	**99%**	**99%**	**99%**	**100%**

Method / Budget	256	512	1024	2048	4096
H2O	2%	2%	2%	2%	4%
TOVA	2%	2%	2%	2%	10%
StreamingLLM	1%	1%	1%	2%	4%
Quest (ours)	**88%**	**92%**	**96%**	**100%**	**100%**

Table 1. (i) Results of 10k length passkey retrieval test on LongChat-7b-v1.5-32k. (ii) Results of 100k length passkey retrieval test on Yarn-Llama-2-7b-128k. Quest can achieve nearly perfect accuracy with a KV cache of 64 and 1024 tokens, which is about 1% of the total sequence length, demonstrating that Quest can effectively preserve the model's ability to handle long-dependency tasks. However, KV cache eviction algorithms such as H2O, TOVA, and StreamingLLM incorrectly discard the KV cache of the answer before receiving the question, thus failing to achieve ideal accuracy.

4.2. Accuracy Evaluation

4.2.1. Language Modeling on PG19

We first evaluate the language modeling perplexity on the PG19 test set, which is a dataset comprising 100 books with an average length of 70k tokens. We use the LongChat-7b-v1.5-32k model to test 32k tokens on PG19. We feed the model with various numbers of tokens and evaluate the perplexity of generated tokens. We evaluate H2O, TOVA, and Quest with a token budget of 4096, which is approximately 1/8 of the total token length. As indicated by the perplexity results in Fig. 6, Quest's accuracy closely matches the oracle baseline with a full KV cache.

4.2.2. Results on long text passkey retrieval task

Since language modeling evaluation only involves local dependencies, models can achieve great performance by fo-

[‡] The top-K operator incurs negligible memory loading and execution time (5-10 us). Therefore, we do not include it in efficiency analysis.

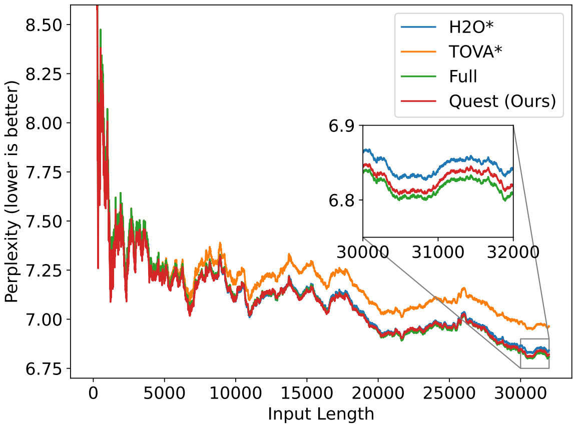


Figure 6. Language modeling evaluation of Quest on PG19 dataset. We prompt the model with 0 to 32000 tokens from the PG19 test set and measure the perplexity of output tokens. H2O* and TOVA* indicate that for the first two layers of models, we do not apply these two algorithms to prune the KV Cache, as analyzed in Sec 3.4, which better preserves the model performance. Quest also uses a full cache in the first two layers of the model. Quest can closely match the performance of the full cache model.

cusing on recent tokens. However, the ability to handle long-distance dependencies is crucial for long text reasoning. For KV cache eviction algorithms like H2O and TOVA, parts of KV caches that are important for distant future tokens may be discarded, thereby preventing the model from obtaining the correct answer. To show that Quest helps maintain the ability of models to handle longer dependency tasks, we evaluate it on the passkey retrieval task from Yarn (Peng et al., 2023). This task measures a model's ability to retrieve a simple passkey from a large amount of meaningless text. We put the answer in different depth ratios of the text and evaluate if the model can retrieve the correct answer with different KV cache token budgets. We evaluate LongChat-7b-v1.5-32k on 10k tokens test and Yarn-Llama-2-7b-128k on 100k tokens test.

Since H2O (Zhang et al., 2023b) needs to calculate historical attention scores for KV cache pruning, it needs to compute the complete $O(n^2)$ attention map and thus is unable to use Flash-Attention (Dao et al., 2022) for long-context inference. Therefore, to enable H2O on long-context evaluation, we use Flash-Attention in the context stage for the 100k sequence length passkey retrieval test and start collecting historical attention scores for H2O in the decoding stage. For TOVA (Oren et al., 2024) and StreamingLLM (Xiao et al., 2023), we evaluated them on the 10k and 100k sequence lengths.

For the passkey retrieval test, we directly prefill the input text containing the passkey and texts to the model. However, to evaluate the impact of different methods on the model's ability to handle long-dependency tasks in practical scenarios, we simulate decoding by feeding the task's question and instruction to the model token by token. In this case, H2O and TOVA might mistakenly discard tokens critical for future tokens, such as the passkey that will be queried later. Similarly, StreamingLLM can only focus on the most recent text window, and if the passkey appears outside this window, it cannot provide the correct answer. Therefore, H2O, TOVA, and StreamingLLM cannot achieve ideal accuracy on the 10k and 100k length passkey retrieve test. However, Quest does not discard KV cache but instead uses a query-aware approach to identify critical tokens. As shown in Tab. 1, Quest can achieve perfect accuracy with a minimal budget both on 10k and 100k sequence length tests.

4.2.3. Results on LongBench

To validate that Quest can outperform baselines on general long-context datasets, we evaluate our method and baselines on six datasets in LongBench. We evaluate on LongChat-7b-v1.5-32k across a wide range of long-context datasets, including single-document QA: NarrativeQA, Qasper, MultiFieldQA; multi-document QA: HotpotQA; summarization: GovReport; few-shot learning: TriviaQA. We evaluate H2O, TOVA, StreamingLLM, and Quest with different KV cache budgets. For all datasets, we split the input into material and question/instruction. For the material part, we use Flash-Attention (Dao et al., 2022) with the full KV cache to perform inference. For the question part, we simulate decoding by feeding them to the model token by token. Similar to the passkey retrieval test, to enable H2O to use Flash-Attention, we could not collect H2O's historical attention scores during the context stage, thus starting from the decoding stage.

As shown in the Fig. 7, Quest consistently outperforms all baselines across six long-context datasets with various KV cache budgets. Quest with a budget of 1K tokens can achieve comparable performance as the model with full KV cache, while other baselines still exhibit a notable gap from full cache performance even with a larger budget. After considering the full cache used in the first two layers, Quest can achieve lossless performance on Qasper, HotpotQA, GovReport, TriviaQA, NarrativeQA, and MultifieldQA with KV cache sparsity of 1/6, 1/6, 1/5, 1/10, 1/5, and 1/6, respectively. This demonstrates that Quest is capable of maintaining the model's capabilities across different types of long-context tasks, as it does not lead to the generation of incorrect answers due to improper discarding of KV cache.

4.3. Efficiency evaluation

To demonstrate the feasibility of Quest, we implement the entire framework with dedicated CUDA kernels based on FlashInfer (Ye et al., 2024), a kernel library for LLM inference. We first evaluate Quest's kernel-level efficiency

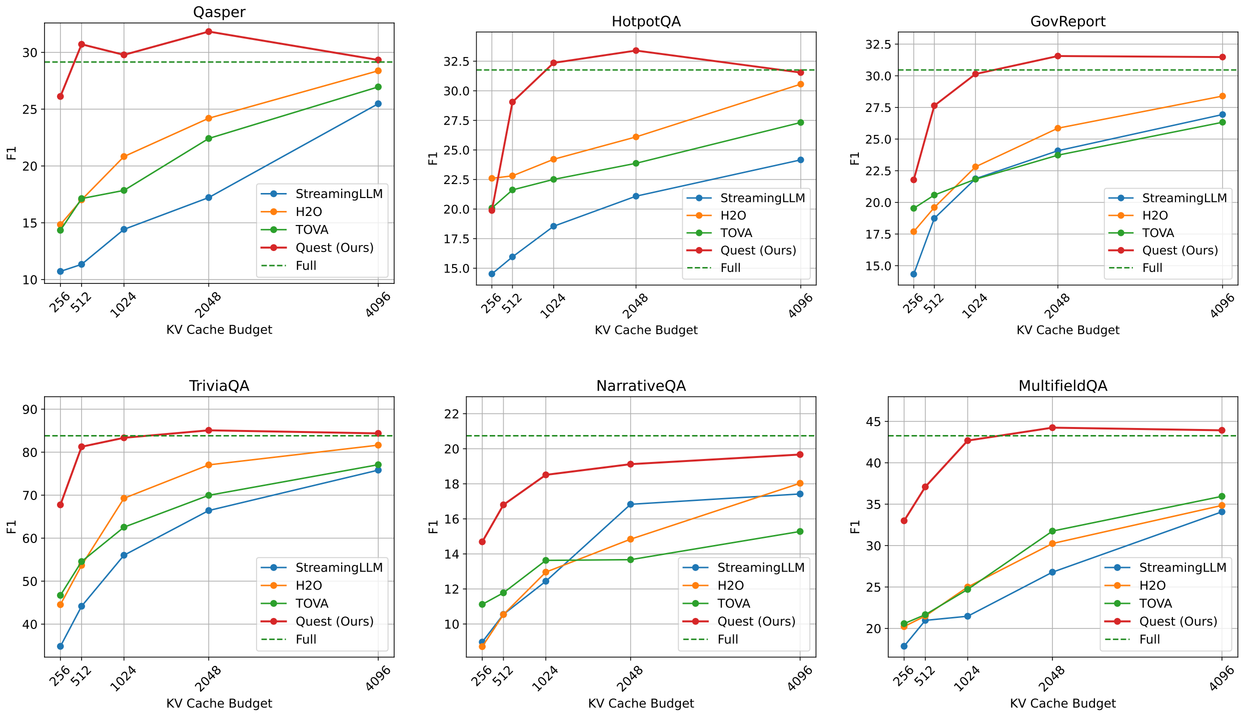


Figure 7. We evaluate Quest across six long context datasets with various token budgets. Quest constantly surpassing all baselines at all datasets and all token budgets. For most of the dataset, Quest reaches comparable accuracy with a 1K token budget.

under the configuration of Llama2-7B on an RTX4090 with CUDA 12.2 in Sec 4.3.1. Besides, we show the end-to-end speedup of Quest in text generation as shown in Sec 4.3.2. We compare Quest with a normal attention implementation from the original FlashInfer. To demonstrate the improvement, we qualitatively compare efficiency under the same accuracy between Quest and baselines in Sec 4.3.3. Note that we use an Ada 6000 GPU (NVIDIA, 2023) in end-to-end evaluations for longer context length.

4.3.1. Kernel evaluation

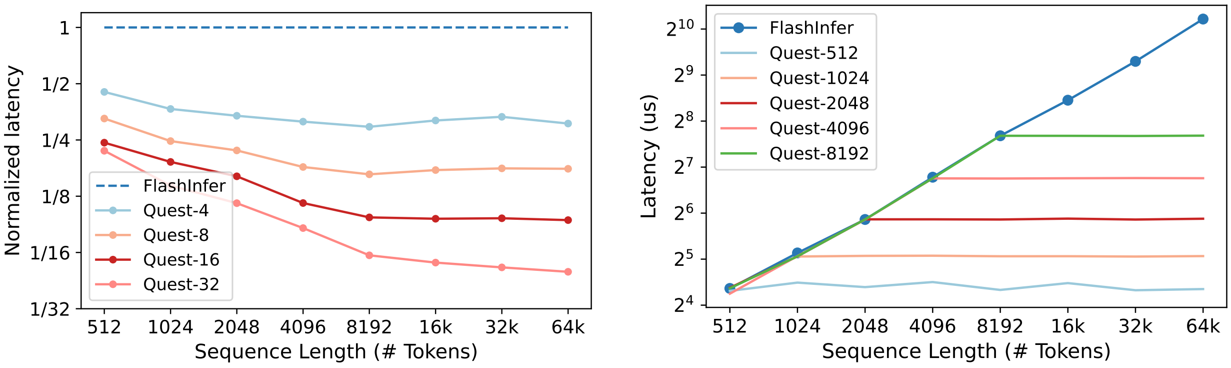
Due to the memory-bound nature of LLM inference, the speedup of Quest is proportional to the sparsity ratio (which is equivalent to memory movement reduction). We quantify this effect in Fig. 8, which evaluates per-kernel performance with NVIDIA's benchmark tool NVBench (NVIDIA, 2024).

Criticality estimation We evaluate the latency of criticality estimation in Quest under different sequence lengths and page sizes. At short sequence length, the memory bandwidth utilization of estimation is smaller than that of FlashInfer, as the total memory load size is not enough to fully utilize GPU memory bandwidth. As sequence length grows, the relative performance improves and approaches $1/\text{Page Size}$ since estimation only consumes one token per page. Note that techniques like quantization or larger page size can further reduce the additional memory usage.

Top-K filtering We enable the Top-K filtering in Quest with a batched Top-K CUDA operator from a vector search kernel library RAFT (Zhang et al., 2023a). We test the latency of Top-K filtering under different sequence lengths and token budgets. Since Criticality estimation reduces one entire token into one criticality score, Top-K filtering has limited memory movement compared to other operators, thus having a low latency overhead of 5-10 us for sequence length less than 128k.

Approximate attention Since Quest is compatible with PageAttention, approximate attention can be easily implemented by feeding Top-K page indices as sparse loading indices. We compare Quest's approximate attention with the original attention of FlashInfer under different sequence lengths and token budgets with a 16 page size. At a given token budget B, the latency of Approximate attention is a constant regardless of the sequence length. Since Approximate attention introduces minimal overhead, it has a similar latency as FlashInfer at sequence length B.

We further evaluate Quest's attention mechanism, which combines Criticality estimation,Top-K filtering, and Approximate attention, on the Llama2-7B model using the PyTorch profiler. We show the time breakdown of Quest in Fig. 9 on various sequence lengths. Quest reduce the self-attention time by $7.03\times$ compared with FlashInfer at 32K sequence length with 2048 token budget.



(a) Criticality estimation

(b) Approximate attention

Figure 8. We measure the latency of individual kernels in Quest. (a) As sequence length increases, the relative criticality estimation latency decreases to $1/\text{Page Size}$ of FlashInfer. (b) Approximate attention with token budget K consumes constant time irrelevant to total sequence length and reaches similar performance of FlashInfer at sequence length K.

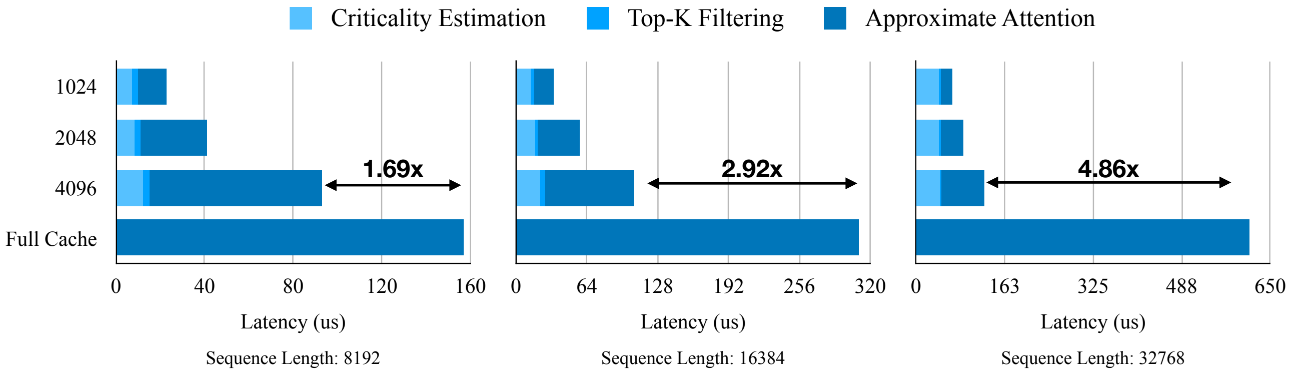


Figure 9. Quest self-attention time breakdown compared to FlashInfer. At all sequence lengths, Quest significantly outperforms FlashInfer, as the memory movement is reduced. At sequence length 32K with token budget 2048, Quest speeds up self-attention by 7.03×.

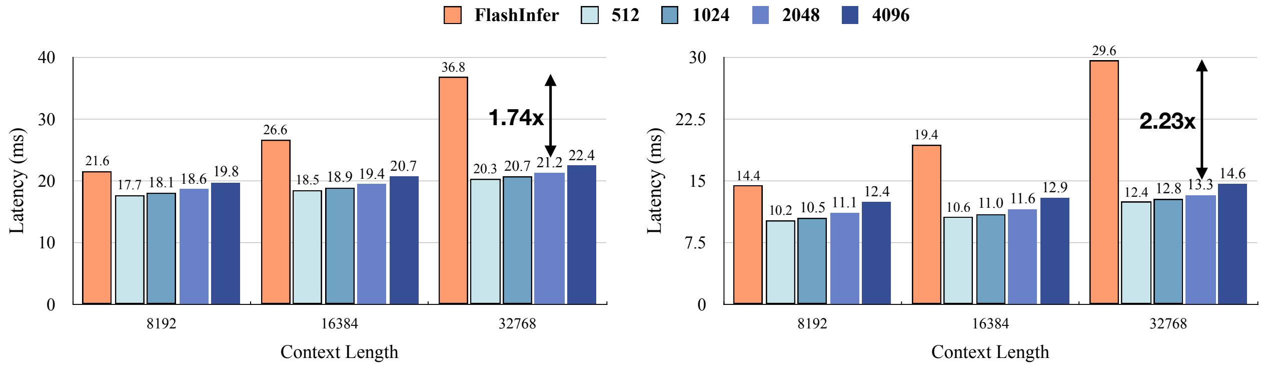
4.3.2. End-to-End Evaluation

To show the practical speedup of Quest, we deploy the framework into real-world single-batch scenarios. We measure the average latency of generating one token in the decode stage under different sequence lengths and token budgets. Note that we do not measure the sampling process since its execution time is smaller and depends on the setting. We compare Quest with a full KV cache baseline which is implemented by FlashInfer. As shown in Fig. 10, Quest outperforms FlashInfer at all sequence lengths. The latency of Quest grows significantly slower than FlashInfer when the sequence length increases, as Quest maintains similar token budgets. At sequence length 32K and token budget 2048, Quest boosts inference speed by 1.74× with FP16 weights and 2.23× with 4-bit quantized weight.

4.3.3. Comparison with Baselines

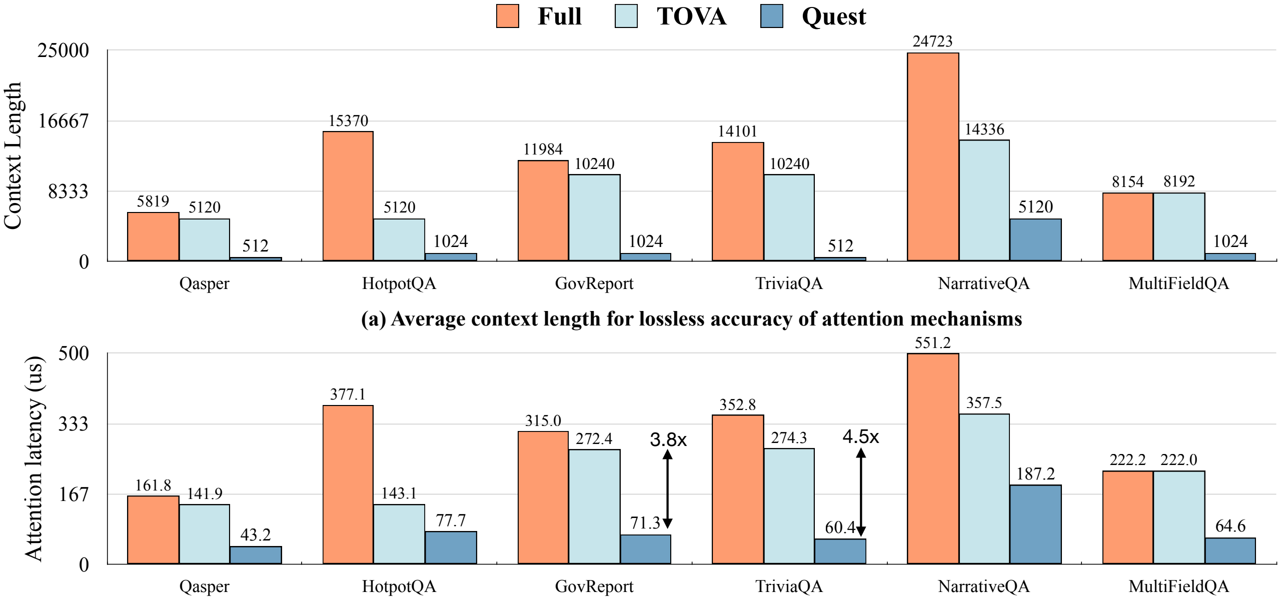
To demonstrate the performance improvements of Quest, we compare the inference efficiency of different attention mechanisms under the same accuracy constraint, i.e. lossless accuracy of six tasks from LongBench. We show token budgets needed for the lossless accuracy target by different attention mechanisms in Fig 11(a). For example, NarrativeQA exhibits an average context length of 24K tokens. To achieve lossless accuracy, TOVA requires a token budget of 14K, whereas Quest necessitates only 5K tokens leading to much higher sparsity.

However, none of the baselines included a kernel implementation of their proposed method. Consequently, we conduct a qualitative analysis of the baselines' self-attention efficiency by utilizing the inference latency of FlashInfer, disregarding other runtime overheads (e.g., TOVA's requirement to calculate history scores (Oren et al., 2024)). In contrast, Quest is evaluated in a practical setting with con-



(a) FP16 Weight **(b) 4-bit Weight (AWQ)**

Figure 10. End-to-end latency of Quest. For all sequence lengths, Quest significantly outperforms FlashInfer. Increasing the sequence lengths only slightly changes the latency of Quest. At a given sequence length, Quest's latency slightly increases as the token budget grows. With sequence length 32K, token budget 2048, 4-bit weight quantization, Quest speedup end-to-end inference by $2.23\times$.



(a) Average context length for lossless accuracy of attention mechanisms

(b) Inference latency of different attention mechanism under benchmarks

Figure 11. Efficiency comparison of Quest with baselines under the same accuracy constraint. (a) Tokens budgets needed for comparable accuracy by different attention methods. Full denotes the original attention, which means the average context length of benchmarks. (b) Inference latency of different attention methods for comparable accuracy. Quest boosts $3.82\times$ speed on GovReport compared to TOVA.

sideration of all operators. As shown in Fig. 11(b), Quest significantly surpasses all baselines in terms of self-attention latency due to the high query-aware sparsity. For GovReport and TriviaQA, Quest boosts the inference by $3.82\times$ and $4.54\times$, respectively. Therefore, Quest can achieve higher efficiency while maintaining superior accuracy.

5. Conclusion

We present Quest, an efficient and accurate KV cache selection algorithm that exploits query-aware sparsity. Quest dynamically estimates the criticality of tokens in KV cache based on the per-page metadata and the current query. It then performs self-attention only on the critical tokens with greatly reduced memory movement, providing high sparsity with negligible accuracy loss. Comprehensive evaluations demonstrate that Quest provides up to $7.03\times$ self-attention speedup, which contributes to $2.23\times$ end-to-end latency reduction in the decode phase. Compared to prior baselines, Quest reduces up to $4.5\times$ self-attention latency with the same accuracy target under long-context benchmarks.

Acknowledgements

We thank Zihao Ye for his insightful discussion, feedback, and useful advice on algorithm design and FlashInfer integration. This work was supported by generous gifts from Intel, Google, and the PRISM Research Center, a JUMP Center cosponsored by SRC and DARPA.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Anthropic. Introducing the next generation of Claude. https://www.anthropic.com/news/claude-3-family, 2024. [Accessed 28-05-2024].

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., Dong, Y., Tang, J., and Li, J. Longbench: A bilingual, multitask benchmark for long context understanding, 2023.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.

Dasigi, P., Lo, K., Beltagy, I., Cohan, A., Smith, N. A., and Gardner, M. A dataset of information-seeking questions and answers anchored in research papers, 2021.

Ge, S., Zhang, Y., Liu, L., Zhang, M., Han, J., and Gao, J. Model tells you what to discard: Adaptive kv cache compression for llms, 2024.

Huang, L., Cao, S., Parulian, N., Ji, H., and Wang, L. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1419–1436, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.112. URL https://aclanthology.org/2021.naacl-main.112.

Joshi, M., Choi, E., Weld, D., and Zettlemoyer, L. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Barzilay, R. and Kan, M.-Y. (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147.

Kočiský, T., Schwarz, J., Blunsom, P., Dyer, C., Hermann, K. M., Melis, G., and Grefenstette, E. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018. doi: 10.1162/tacl_a_00023. URL https://aclanthology.org/Q18-1023.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Li, D., Shao, R., Xie, A., Sheng, Y., Zheng, L., Gonzalez, J. E., Stoica, I., Ma, X., and Zhang, H. How long can open-source llms truly promise on context length?, June 2023. URL https://lmsys.org/blog/2023-06-29-longchat.

Liu, H., Yan, W., Zaharia, M., and Abbeel, P. World model on million-length video and language with blockwise ringattention, 2024a.

Liu, X., Yan, H., Zhang, S., An, C., Qiu, X., and Lin, D. Scaling laws of rope-based extrapolation, 2024b.

NVIDIA. Nvidia ada lovelace professional gpu architecture. https://images.nvidia.com/aem-dam/en-zz/Solutions/technologies/NVIDIA-ADA-GPU-PROVIZ-Architecture-Whitepaper_1.1.pdf, 2023. [Accessed 28-05-2024].

NVIDIA. Nvbench: Nvidia's benchmarking tool for gpus, 2024. Available online: https://github.com/NVIDIA/nvbench.

OpenAI. New models and developer products announced at devday. https://openai.com/blog/new-models-and-developer-products-announced-at-devday#OpenAI, November 2023. Accessed: 2024-01-31.

OpenAI. Introducing gpt-4o: our fastest and most affordable flagship model. https://platform.openai.com/docs/models, 2024. [Accessed 28-05-2024].

Oren, M., Hassid, M., Adi, Y., and Schwartz, R. Transformers are multi-state RNNs, 2024. URL https://arxiv.org/abs/2401.06104. arXiv:2401.06104.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. Yarn: Efficient context window extension of large language models, 2023.

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. *arXiv preprint*, 2019. URL https://arxiv.org/abs/1911.05507.

Ribar, L., Chelombiev, I., Hudlass-Galley, L., Blake, C., Luschi, C., and Orr, D. Sparq attention: Bandwidth-efficient llm inference, 2023.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023.

Tworkowski, S., Staniszewski, K., Pacek, M., Wu, Y., Michalewski, H., and Miłoś, P. Focused transformer: Contrastive training for context scaling, 2023.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *arXiv*, 2023.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018.

Ye, Z., Lai, R., Lu, R., Lin, C.-Y., Zheng, S., Chen, L., Chen, T., and Ceze, L. Cascade inference: Memory bandwidth efficient shared prefix batch decoding. https://flashinfer.ai/2024/01/08/cascade-inference.html, Jan 2024. URL https://flashinfer.ai/2024/01/08/cascade-inference.html. Accessed on 2024-02-01.

Zhang, J., Naruse, A., Li, X., and Wang, Y. Parallel top-k algorithms on gpu: A comprehensive study and new methods. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, New York, NY, USA, 2023a. Association for Computing Machinery. ISBN 9798400701092. doi: 10.1145/3581784.3607062. URL https://doi.org/10.1145/3581784.3607062.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., Wang, Z., and Chen, B. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023b.

Zhao, Y., Lin, C.-Y., Zhu, K., Ye, Z., Chen, L., Zheng, S., Ceze, L., Krishnamurthy, A., Chen, T., and Kasikci, B. Atom: Low-bit quantization for efficient and accurate llm serving, 2024.