

Memory-Efficient KV Cache Optimization for Large Language Model Inference at the Edge

Chi Zhang^{*†‡}, Haisheng Tan[†], Haotian Pan[†], Yang Xu[‡], Haohua Du[§], Li Zhang^{*}, and Xiaoming Fu[†]

^{*}Hefei University of Technology, China [†]University of Göttingen, Germany

[‡]University of Science and Technology of China, China [§]Beihang University, China

Email: zhangchi@hfut.edu.cn, hstan@ustc.edu.cn, {haotianpan, xu_yang}@mail.ustc.edu.cn, duhaohua@buaa.edu.cn, lizhang@hfut.edu.cn, fu@cs.uni-goettingen.de

Abstract—The deployment of large language models (LLMs) on edge devices offers notable advantages in privacy preservation, low-latency response, and operational autonomy, but faces significant challenges due to the memory-intensive key-value (KV) cache required for autoregressive decoding. Sparse attention methods alleviate computational costs but typically fail to preserve the generation quality and inference efficiency under memory constraints, which becomes particularly acute on edge platforms. In this paper, we first formalize and prove the NP-hardness of the KV cache replacement problem under top- k sparse attention, where k is the number of pages involved in each attention calculation. Then, we propose **LiteKV**, a lightweight memory-efficient online KV cache replacement algorithm tailored for edge-based LLM inference. **LiteKV** introduces a $(1 + \epsilon)$ -relaxation strategy that tolerates minimal precision loss in attention scores, allowing significant improvements in cache utilization. Moreover, **LiteKV** is orthogonal to existing sparse attention and quantization methods, enabling scalable, high-performance LLM inference under stringent memory budgets. We provide theoretical guarantees on **LiteKV**'s competitive ratio under realistic attention score distributions. Experiments on real-world workloads demonstrate that **LiteKV** reduces memory usage by up to 75% compared to state-of-the-art memory-unconstrained baselines, while preserving comparable generation quality and 95% of their decoding efficiency.

I. INTRODUCTION

Large Language Models (LLMs) have significantly advanced natural language processing through capabilities in coherent text generation, sophisticated instruction understanding, and few-shot learning, enabling various powerful applications. However, the substantial computational and memory requirements of LLMs have typically restricted their deployment to cloud infrastructures. As intelligent applications progressively migrate to edge computing environments, where low latency, real-time processing, and data privacy are paramount, on-device LLM inference becomes essential, particularly for latency-sensitive applications like autonomous vehicles, smart wearables, and privacy-sensitive interactions.

A fundamental challenge in deploying LLMs on edge devices lies in the memory-intensive key-value (KV) cache required for autoregressive generation. Its storage demand scales proportionally with context length and quickly overwhelms the limited memory capacities of edge devices. Early sparse attention mechanisms, such as H2O [1], StreamingLLM [2], and SnapKV [3], reduced overhead by evicting less relevant KV pages, but their query-agnostic nature leads to quality degrada-

tion for long contexts. More recent approaches like Quest [4] improve efficiency via top- k token selection in attention, but they still assume unlimited GPU memory, which is unrealistic in edge settings. This creates a fundamental tension: how to balance memory efficiency with generation fidelity under strict memory budgets? We identify a critical gap: **existing methods optimize attention computation but neglect the cache replacement problem under memory constraints**. In this work, we focus on strategic KV cache management, which preserves only the most generation-critical KV pairs while enabling efficient, high-quality inference on edge devices.

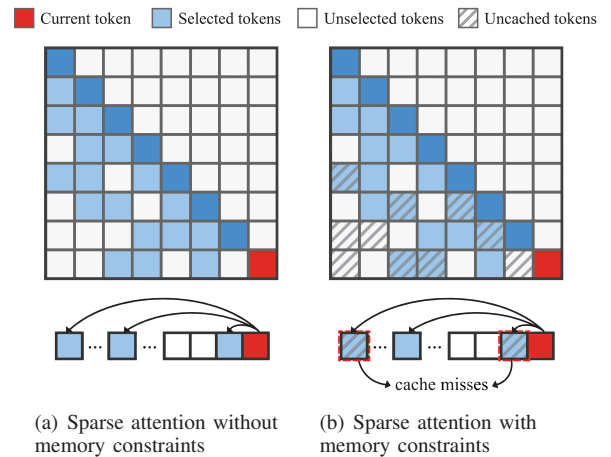


Fig. 1. Illustration of sparse attention with memory constraints, where we need to determine both the pages involved in attention computation and the ones to be cached to minimize cache misses while retaining generation quality.

Illustrative Example. Fig. 1 illustrates the critical distinction between sparse attention with and without memory constraints, where dark blue grids represent the tokens generated in previous iterations. The red represents the token currently generated, with dependency arrows connecting to its influential context tokens (light blue grids in the same row). As shown in Fig. 1(a), we first demonstrate the KV states of standard sparse attention methods [4]–[8], where only a subset of previous KV pages are selected at each decoding step to participate in the attention computation, thereby reducing the overall computational cost. Specifically, these methods implicitly rely on unlimited GPU memory, an assumption that is violated in edge deployments as retaining all pages of previous tokens

becomes infeasible. Fig. 1(b) depicts the situation of sparse attention with memory constraints, where the grey hatched grids represent pages evicted from GPU memory. When a token's attention computation requires missing pages (e.g., two light blue grids overlapping with grey hatch), cache misses occur, leading to costly retrieval via transfer or recomputation. As the context length grows, the fixed memory budget necessitates an increasing number of pages to be evicted from the GPU memory, intensifying the fundamental conflict between memory constraints and long-context inference.

In this work, we aim to address two core problems: 1) with consideration of memory budgets, selecting which pages to include in attention computation so as to balance decoding efficiency and token generation quality, i.e., determining which grids are in light blue as shown in Fig. 1, and 2) selecting which pages to be retained through replacement strategies that adapt to their evolving importance during the decoding process, i.e., determining which grids are not hatched in grey.

To tackle these challenges, we propose a lightweight KV cache replacement algorithm tailored for edge environments, called *LiteKV*. By jointly considering attention precision and cache efficiency under memory constraints, *LiteKV* achieves both faster decoding and consistent generation quality. We make the following contributions:

- In order to characterize the KV cache replacement problem under strict memory constraints in edge scenarios, we formulate the KV cache replacement problem within top- k sparse attention and prove its NP-hardness, where k is the number of pages involved in each attention calculation.
- We propose *LiteKV*, a lightweight, online cache replacement algorithm that leverages a $(1 + \epsilon)$ -relaxation of attention scores to trade minimal precision loss for dramatically improved cache efficiency. *LiteKV* adapts classical caching strategies MIN [9] and LRU [10] to the top- k setting and provides a theoretical competitive ratio under realistic attention distributions. Notably, *LiteKV* is orthogonal to sparse attention and quantization, enabling seamless integration into existing edge-optimized LLM pipelines.
- We implement *LiteKV* on several widely adopted long-context LLMs. Extensive experiments on real-world workloads demonstrate that *LiteKV* reduces memory usage by up to 75%, preserves near-identical generation quality, and achieves up to 54.84% lower decoding latency compared to state-of-the-art baselines under identical memory constraints.

Our work highlights that relaxed token selection is not a compromise but a strategic enabler for practical edge deployment of LLMs.

II. MOTIVATION

A. The Quality-Memory-Efficiency Trilemma

While recent research has focused on improving LLM inference efficiency via sparse attention and quantization, these methods primarily target compute reduction or model compression, often neglecting the critical challenge of GPU

memory constraints during decoding. Some early works, such as H2O [1], StreamingLLM [2], SnapKV [3], etc., address memory constraints by permanently discarding KV pages, which can significantly impact generation quality in long-context scenarios [4]. On the other hand, query-aware sparse attention methods such as Quest [4] retain all previously generated KVs during autoregressive generation to preserve accuracy, leading to unbounded memory growth as the sequence length increases. As illustrated in Fig. 2, even with pooling and quantization applied, the KV cache memory usage in Quest still scales linearly with the context length. For instance, even for a lightweight model like LLAMA2-7B, the KV cache of Quest consumes up to 16GB of GPU memory when the context length reaches approximately 30K tokens. This ever-expanding memory footprint poses a significant challenge in resource-constrained environments, where neither KV offloading to external memory nor GPU capacity expansion represents a feasible solution. In short, current sparse attention methods fail to preserve the generation quality and inference efficiency with constrained memory, which becomes particularly acute in edge platforms.

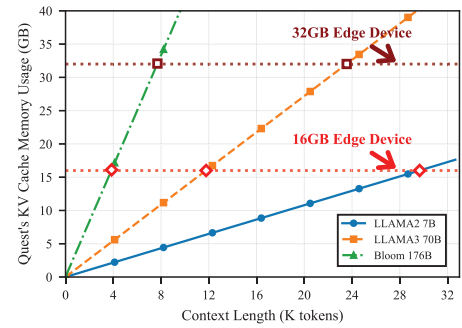


Fig. 2. Quest's KV cache memory usage with increasing context length.

Unlike data center deployments that can maintain complete intermediate KV states, edge inference systems must make judicious decisions about KV cache replacement under strict memory budgets. To fully utilize the limited on-device memory, it is essential to develop cache replacement strategies that determine which KV pages are most valuable for ongoing inference and should therefore be retained in memory. This motivates our investigation into KV cache management for memory-constrained edge-based LLM deployment.

B. Strict Top- k Selection is not Cache-Efficient

In memory-constrained environments, efficient KV cache management becomes pivotal for long-context LLM inference. While strict top- k sparse attention mechanisms (e.g., H2O and Quest), where only the k most relevant previous tokens are selected for attention computation, effectively reduce computational overhead, their inherent memory-unaware design results in suboptimal cache utilization. Specifically, these approaches frequently incur cache misses even when employing conventional replacement policies like LRU.

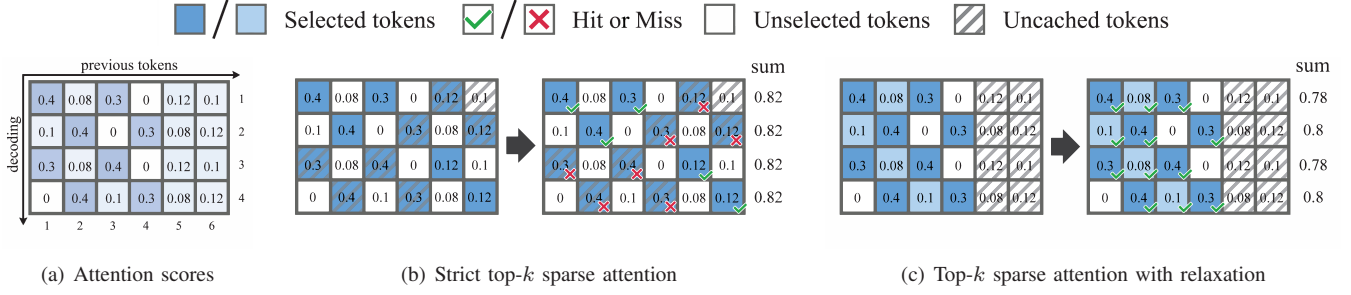


Fig. 3. Motivating example to show that strict top-k sparse attention is not cache-efficient, where cache size $C = 4$ and $k = 3$.

Motivating Example. Fig.3 illustrates a scenario with cache size $C = 4$ and $k = 3$, where the first 4 tokens are initially cached. We simulate 4 token generations, each attending to a fixed set of 6 previous tokens as shown in Fig. 3(a). We first demonstrate strict top-k sparse attention with LRU policy, as depicted in Fig. 3(b), where only the k highest-scoring pages are selected for attention computation, and the least recently used KVs will be evicted to satisfy memory constraints. While strict top-k achieves an average attention score of 0.82, it delivers a cache hit ratio of only 42% (5 hits out of 12 KV accesses), triggering either redundant recomputation or I/O consumption. In contrast, Fig. 3(c) presents a relaxed alternative where a minor concession in attention precision, retaining approximately 95% of the total attention score compared to strict top-k, allows the cache hit ratio to reach 100%. This shows that a small trade-off in precision leads to significantly better cache efficiency under tight memory budgets.

This example underscores that, without joint consideration of attention computation and cache replacement under memory constraints, decoding efficiency will be severely compromised. Solving this problem is not merely an optimization, but a prerequisite for practical LLM inference at the edge.

III. PROBLEM FORMULATION

A. Sparse Attention

Consistent with previous research [8], we consider sparse attention with query $Q \in \mathbb{R}^{1 \times d}$ and key-value matrices $K, V \in \mathbb{R}^{l \times d}$, where d is the head dimension and l is the current context length. The selected token indices form a subset $\mathcal{I}$. The attention output is:

$$\hat{O} = \text{softmax} \left(\frac{Q \cdot K^T}{\sqrt{d}} \right) \Lambda_{\mathcal{I}} V = W \Lambda_{\mathcal{I}} V, \quad (1)$$

where $\Lambda_{\mathcal{I}} \in \mathbb{R}^{l \times l}$ is a diagonal matrix with entries:

$$\Lambda_{\mathcal{I}}[i, j] = \begin{cases} 1 & \text{if } i = j \text{ and } i \in \mathcal{I}, \\ 0 & \text{otherwise.} \end{cases}$$

To minimize the output error $\|\hat{O} - O\|$, we need to carefully select the subset of tokens for attention computation. Previous research [11] suggests a tight error bound given by:

$$\mathcal{L} = \|\hat{O} - O\| = \|W(\Lambda_{\mathcal{I}} - \mathbf{1}^{l \times l})V\| \leq \|W(\Lambda_{\mathcal{I}} - \mathbf{1}^{l \times l})\| \cdot \|V\|. \quad (2)$$

The objective reduces to minimizing $\|W(\Lambda_{\mathcal{I}} - \mathbf{1}^{l \times l})\| = 1 - \sum_{i \in \mathcal{I}} W[i]$, equivalent to selecting a subset of tokens that maximizes the sum of attention weights. For a fixed subset size $|\mathcal{I}| = k$, the oracle top-k sparse attention is defined as:

$$\mathcal{I} = \arg \max_{\mathcal{I}} \sum_{i=1}^l W \Lambda_{\mathcal{I}} \quad \text{s.t.} \quad |\mathcal{I}| = k. \quad (3)$$

B. KV Cache Replacement

Let n denote the total context length throughout the decoding phase, also known as the number of KV pages. Let m represent the total number of tokens to be generated, also referred to as requests. Denote $s_{i,j}$ as the attention score $W[i]$ of the i -th KV page for the j -th request, where $1 \leq i \leq n$. The sum of scores across all KV pages is normalized to 1, i.e., $\sum_{i=1}^n s_{i,j} = 1$.

Let C denote the capacity of GPU memory, i.e., at most C pages can be stored in GPU memory. When a request requires a page not present in the GPU memory, a cache miss occurs, necessitating the transfer or recomputation of the missed page. In this context, GPU memory serves as a cache. Unless specified as “KV cache”, the term “cache” hereafter refers to GPU memory. Let $\mathcal{S}_j$ denote the cache content set of KV pages before the j -th request, we have $|\mathcal{S}_j| \leq C, \forall j$. The objective to minimize the total cache misses is $\min \sum_j \sum_{i \in \mathcal{I}_j} I(i \notin \mathcal{S}_j)$, where $I(i \notin \mathcal{S}_j)$ is an indicator that equals 1 if $i \notin \mathcal{S}_j$ and 0 otherwise. In the rest of the paper, we refer to it as the top-k problem if $\mathcal{I}_j$ satisfy Eq. (3).

C. Problem Hardness

We first prove the NP-hardness of the offline top-k problem.

Theorem 1. *The offline top-k problem is NP-hard.*

Proof. We first consider a simplified case by setting $k = 1$ and assuming an unbounded cache size ($C \rightarrow +\infty$). For each request, every page is assigned a score of either 0 or a uniform constant value.

Consider an instance x of the set cover problem [12], defined by a universe $U = \{1, 2, \dots, m\}$ and n subsets $S_1, S_2, \dots, S_n \subseteq U$. And consider an instance y of the simplified top-k problem, comprising m requests and n pages. We construct a polynomial-time reduction from instance x to y . For each element $j \in S_i$ in x , the score $s_{i,j}$ in y is assigned a non-zero value; otherwise $s_{i,j} = 0$. Consequently,

the minimum number of sets required to cover U corresponds to the minimum cache load cost to satisfy all requests. Thus, we establish that set cover is polynomial-time reducible to a special case of top- k problem. Since set cover is NP-hard, it follows that the top- k problem is also NP-hard. $\square$

Then, for the online version, we demonstrate that no efficient online algorithm exists for the top- k problem.

Theorem 2. *No online algorithm can achieve a finite competitive ratio for the top- k problem.*

Proof. In analyzing the top- k problem, we consider the special case where $C = k = 1$. We demonstrate that the competitive ratio is unbounded even in this minimal case. For the first request, all page scores are set equal. When the online algorithm selects a page, its score is set to zero for the subsequent request. Consequently, selecting a page with a non-negative score results in its score being zeroed for the next request, leading to m cache misses over m requests. In contrast, as the number of pages exceeds the number of requests, there exists a page with a consistently non-negative score. The offline optimal selects this page initially, incurring a cost of 1. Thus, the competitive ratio for this case is m , which grows unbounded as $m \rightarrow \infty$. $\square$

Our analysis reveals that the hardness of the top- k problem primarily arises from a large number of identical page scores. However, experimental observations indicate that the probability of encountering a large number of identical page scores is extremely low. Consequently, in the rest of this paper, we assume that all page scores are distinct for a certain request. Under this assumption, we find that for the top- k problem, the pages associated with each request are uniquely determined. When the cache size C is sufficiently large, the offline top- k problem is clearly in the complexity class P. We further demonstrate in subsequent sections that, even with a finite C , the offline top- k problem with distinct page scores remains in P. This assumption also enables us to establish bounds on the competitive ratio in online settings.

D. $(1 + \epsilon)$ -Relaxation

To address the top- k problem under the above assumption, we then introduce the concept of $(1 + \epsilon)$ -relaxation to handle closely aligned page scores. Based on the definition of top- k sparse attention presented in Eq. (3), we introduce the $(1 + \epsilon)$ -relaxation attention as:

Definition 1. An attention is defined as a $(1 + \epsilon)$ -relaxation if and only if the indices of selected tokens $\mathcal{I}'$ satisfy $|\mathcal{I}' \setminus \mathcal{I}| = |\mathcal{I}' \setminus \mathcal{I}|$, and there exists an $l \times l$ permutation matrix P such that $W\Lambda_{\mathcal{I}' \setminus \mathcal{I}} \leq W\Lambda_{\mathcal{I} \setminus \mathcal{I}'}P \leq (1 + \epsilon)W\Lambda_{\mathcal{I}' \setminus \mathcal{I}}$, where ϵ is a positive constant.

For the j -th request, if a top- k page i incurs a miss, the $(1 + \epsilon)$ -relaxation allows substitution with a non-top- k page i' in GPU memory, provided that $s_{i,j} \leq (1 + \epsilon)s_{i',j}$.

To evaluate the performance of algorithms employing this relaxed policy, we introduce the following definition for competitive analysis.

Definition 2. An online algorithm is $(1 + \epsilon)$ -relaxation c -competitive if for any request sequence, the ratio of cache misses satisfies $\frac{\text{ALG}}{\text{OPT}} \leq c$, where ALG represents the cache misses incurred by the online algorithm with $(1 + \epsilon)$ -relaxation and OPT denotes the total cache misses of the offline optimal solution without $(1 + \epsilon)$ -relaxation.

IV. ALGORITHM DESIGN

In this section, we first introduce an exact algorithm TopK-MIN to compute an optimal solution for the offline top- k problem. To enable its deployment in real-world experimental settings, we propose an online algorithm, TopK-LRU, which builds on the principles of TopK-MIN. By incorporating a $(1 + \epsilon)$ -relaxation, we further refine TopK-LRU to develop LiteKV, to further improve cache hit ratio and thereby the overall decoding efficiency.

A. Offline Algorithm TopK-MIN

Algorithm 1: Offline Exact Algorithm TopK-MIN

```

1 Input Model parameter  $k$ , cache size  $C$ , page number
    $n$ , request number  $m$ , attention scores  $s_{i,j}$ 
2 Initialize cache content set  $\mathcal{S}$ , top- $k$  sets
    $(\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_m)$  and ordered lists  $(\mathcal{O}_1, \mathcal{O}_2, \dots, \mathcal{O}_n)$ 
   as empty;
3 for  $i \in [n]$ ,  $j \in [m]$  do
4   if  $s_{i,j}$  is top- $k$  of request  $j$  then
5      $\mathcal{I}_j \leftarrow \mathcal{I}_j \cup \{i\}$ ;
6     Append  $j$  to  $\mathcal{O}_i$ ;
7 for  $j \in [m]$  do
8   for  $q \in \mathcal{I}_j \setminus \mathcal{S}$  do
9     if  $|\mathcal{S}| < C$  then
10       $\mathcal{S} \leftarrow \mathcal{S} \cup \{q\}$ ;
11     else
12      Select a page  $r \in \mathcal{S} \setminus \mathcal{I}_j$  that maximizes the
        lexicographical order of  $\mathcal{O}_r$ ;
13       $\mathcal{S} \leftarrow \mathcal{S} \cup \{q\} \setminus \{r\}$ ;
14   Deleting  $j$  across  $\{\mathcal{O}_i | i \in [n]\}$ ;
```

We propose a polynomial-time exact algorithm, named TopK-MIN, for the offline top- k problem without $(1 + \epsilon)$ -relaxation. TopK-MIN follows the offline optimal algorithm MIN. It extends the MIN approach by comparing pages based on their complete request sequences, as outlined in Algorithm 1. The algorithm starts with data structure initialization (Line 2). Next, it constructs the top- k sets and ordered lists by iterating over all pages (Lines 3-6). The core processing phase then sequentially handles missed pages for each request (Lines 7-14). If cache space is available, pages are added directly (Lines 9-10). Otherwise, TopK-MIN selects a page

from the unvisited set with the maximum lexicographical order (indicating the furthest future use), and replaces it with the incoming page (Lines 11-13). After processing each request, the algorithm updates future request information to prioritize pages for subsequent requests (Line 14).

Theorem 3. *TopK-MIN is an optimal algorithm for the offline top-k problem with distinct attention scores.*

Proof. The optimality of TopK-MIN can be established by extending the proof of MIN's optimality [13]. Due to space limitations, we omit the details here. $\square$

B. Online Algorithm TopK-LRU

The classical LRU emulates the offline MIN algorithm, achieving a competitive ratio C . Likewise, we enhance LRU by incorporating the principles of TopK-MIN, resulting in an online algorithm called TopK-LRU. This algorithm is invoked when the cache reaches full capacity, following the greedy loading of the top- k pages from initial requests into the cache.

Algorithm 2: Online Algorithm TopK-LRU

```

1 Input Model parameter  $k$ , cache size  $C$ , cache content
   ordered list  $S$ 
2 if arrival of a request then
3   Compute top- $k$  set  $\mathcal{I}$  of the request;
4   for  $q \in \mathcal{I}$  do
5     if  $q \in S$  then
6       Move  $\{q\}$  to head of  $S$ ;
7     else
8       Select page  $r$  at tail of  $S \setminus \mathcal{I}$ ;
9        $S \leftarrow S \cup \{q\} \setminus \{r\}$ ;

```

The primary distinction between TopK-LRU and LRU lies in the protection of the top- k pages from replacement due to concurrent requests. As shown in Algorithm 2, upon receiving a request, TopK-LRU identifies the top- k pages (Lines 2-3). It then checks each page's presence in the cache. If present, the page is promoted to the head of the list (Lines 5-6); if absent, it evicts a non-top- k page from the tail of the list, replacing it with the incoming page (Lines 7-9). This mechanism ensures the cache prioritizes recently accessed pages while safeguarding the top- k pages from eviction.

Theorem 4. *TopK-LRU is C -competitive for the online top-k problem with distinct attention scores.*

Proof. The proof of the competitive ratio of TopK-LRU utilizes a phase-based analysis, similar to that for LRU [14]. The request sequence is divided into phases as follows: Consider a cache S with capacity C . The sequence is processed sequentially, beginning with phase 1. For request j and phase i , if all requested pages can be stored in S without eviction, they are inserted into S , and request j is assigned to phase i . Otherwise, S is cleared, and a new phase $i + 1$ begins with request j . Each phase thus contains requests for at most C

distinct pages. By the phase definition, when a new phase begins, the optimal offline algorithm incurs between 1 and k cache misses. In contrast, TopK-LRU incurs at most C misses per phase. Consequently, TopK-LRU is C -competitive. $\square$

C. Online Algorithm LiteKV

Algorithm 3: Online Algorithm LiteKV

```

1 Input Model parameter  $k$ , cache size  $C$ , cache content
   ordered list  $S$ 
2 if arrival of a request then
3   Compute top- $k$  set  $\mathcal{I}$  for the request;
4   Set of candidate pages  $\mathcal{N} \leftarrow S \setminus \mathcal{I}$ ;
5   for  $q \in \mathcal{I}$  do
6     if  $q \in S$  then
7       Move  $\{q\}$  to head of  $S$ ;
8     else
9       Set of alternative pages
        $U \leftarrow \{i \in \mathcal{N} | s_q \leq (1 + \epsilon)s_i\}$ ;
10      if  $U \neq \emptyset$  then
11        Select page  $r \leftarrow \arg \min_i \{s_i | i \in U\}$ ;
12         $\mathcal{N} \leftarrow \mathcal{N} \setminus \{r\}$ ;
13        Replace  $q$  with  $r$  for calculation;
14      else
15        Select page  $r$  at tail of  $S \setminus \mathcal{I}$ ;
16         $S \leftarrow S \cup \{q\} \setminus \{r\}$ ;

```

The online algorithm LiteKV builds upon TopK-LRU, facilitating efficient cache replacement through dynamic adaptation to incoming requests. Its core mechanism utilizes $(1+\epsilon)$ -relaxation: when a requested page is absent from the cache, the algorithm prioritizes an alternative cached page with a marginally lower score, controlled by the parameter ϵ .

Algorithm 3 outlines the implementation of LiteKV, which closely aligns with TopK-LRU. Initially, upon a request's arrival, it computes the top- k set and identifies potential alternative pages (Lines 3-4). If the requested page is already in the cache, the algorithm updates its position to the head to reflect recent access (Line 7). The key difference from TopK-LRU lies in the $(1+\epsilon)$ -relaxation process (Lines 9-13). For an absent page, the algorithm constructs a set of alternative pages with scores within $(1+\epsilon)$ times that of the absent page (Line 9). If such candidates exist, the algorithm selects the page with the smallest score for replacement and removes it from the candidate set (Lines 10-13). If no candidates are found, it evicts the least recently used page and inserts the incoming page, consistent with TopK-LRU (Lines 15-16). This approach substantially enhances cache efficiency while preserving cache relevance and recency.

Based on observations of attention scores in real-world LLMs, we find that the score distribution is relatively concentrated and exhibits distinguishable patterns. Therefore, we introduce the following definition to facilitate subsequent theoretical analysis of LiteKV.

Definition 3. A sequence $\{a_i\}_{i=1}^n$ is defined as (x, y, z) -close, where $x < y$ and $x + y < n - 1$, if it meets the condition: let A_i denote the i -th largest element of $\{a_i\}_{i=1}^n$, then for $i \in [1, x]$, $A_i < \frac{1}{z^i}$, and for $i \in [n - y + 1, n - y + x]$, $A_i > \frac{1}{z^i}$.

Empirical evidence from extensive real-world experiments demonstrates the existence of a constant B such that the sequence of scores across all pages is (k, C, B) -close for any request. Therefore, we have the following theorem:

Theorem 5. For $\epsilon > B^{n+1-C-k} - 1$, LiteKV is $(1 + \epsilon)$ -relaxation $(k - \lfloor \frac{\log(1+\epsilon) + (C+k-n)\log B}{\log B} \rfloor) \frac{C}{k}$ -competitive for the online top- k problem with distinct attention scores, provided the scores meet (k, C, B) -close property.

Proof. We establish our proof by analyzing the worst cases, where pages in the cache consistently receive the lowest scores. To achieve a competitive ratio superior to TopK-LRU, it is essential to ensure that a subset of the top- k set can be replaced, even under worst-case conditions.

We examine the relationship between the top- k pages among all pages, denoted as $\mathcal{I}$, and the top- k pages in the cache, denoted as $\mathcal{J}$. Consider all scores of n pages sorted into a sequence $\{s_i\}_{i=1}^n$, where s_i represents the i -th largest score. Here, $\mathcal{I}$ corresponds to scores s_1 through s_k , while $\mathcal{J}$ corresponds to scores s_{n-C+1} through s_{n-C+k} , as cached pages are assigned the lowest scores.

Given that $\{s_i\}_{i=1}^n$ satisfies the (k, C, B) -close property, the score differences between pages in $\mathcal{I}$ and $\mathcal{J}$ can be bounded by an exponentially decaying function.

If N ($N \leq k$) pages in $\mathcal{I}$ can be replaced by pages in $\mathcal{J}$, the smallest N pages in $\mathcal{I}$ are always replaceable by the largest N pages in $\mathcal{J}$. Given $s_k < 1/B^k$ and $s_{n-C+N} > 1/B^{n-C+N}$, if $B^{n-C+N}/B^k < 1 + \epsilon$, or equivalently, $\epsilon > B^{n+N-C-k} - 1$, then $s_k/s_{n-C+N+1} < 1 + \epsilon$. Furthermore, the inequality $s_{k+1-i}/s_{n-C+N+1-i} < 1 + \epsilon$ holds for all $i \in [1, N]$. Thus, the condition $\epsilon > B^{n+N-C-k} - 1$ is equivalent to the replaceability of N pages from $\mathcal{I}$ by pages in $\mathcal{J}$.

When $N \geq 1$ (i.e., $\epsilon > B^{n+1-C-k} - 1$), let N_m denote the maximum number of replaceable pages in $\mathcal{I}$. Then, $\epsilon > B^{n+N_m-C-k} - 1$, and solving for N_m yields:

$$N_m = \left\lfloor \frac{\log(1 + \epsilon) + (C + k - n) \log B}{\log B} \right\rfloor. \quad (4)$$

Thus, in the worst case, provided $\epsilon > B^{n+1-C-k} - 1$, at least N_m pages in $\mathcal{I}$ can be replaced, reducing the number of required pages from k to $k - N_m$. This results in a ratio of $(k - N_m)/k$. Subsequent proofs can be derived based on Theorem 4. We divide the request sequence into phases, as defined in the proof of Theorem 4. Within each phase, TopK-LRU incurs up to C cache misses, while LiteKV causes at most $(k - N_m)C/k$ misses. The offline optimal makes at least one miss each time a new phase begins. Thus, the competitive ratio is bounded by $(k - N_m)C/k$, that is:

$$\left(k - \left\lfloor \frac{\log(1 + \epsilon) + (C + k - n) \log B}{\log B} \right\rfloor \right) \frac{C}{k}. \quad \square$$

V. EVALUATION

We prototyped our algorithm LiteKV based on the Quest framework. To evaluate the quality of the token generation, we measured perplexity and passkey retrieval ratio of LiteKV. We further compared the GPU High Bandwidth Memory (HBM) utilization of our proposed approach with existing methods. Lastly, we assessed the decoding efficiency achieved by LiteKV. Our key findings are summarized as follows:

- LiteKV achieves a memory reduction of 75% compared with state-of-the-art sparse attention frameworks, without compromising token generation quality, while still maintaining 95% of their decoding efficiency.
- LiteKV reduces decoding latency by up to 54.84%, attributed to a cache miss ratio reduction of 17.76%, especially under more memory-constrained settings, where efficient KV reuse becomes increasingly critical.
- LiteKV maintains stable decoding latency as the context length grows, indicating its strong scalability in long-context inference scenarios.

A. Experimental Settings

Benchmarks and Models. We test LiteKV using the following benchmarks: PG19 [15], LongBench [16], and the passkey retrieval task [17]. Specifically, we use PG19 to measure the perplexity of the generated text, and employ various datasets from LongBench to assess the cache hit ratio. In addition, we perform the passkey retrieval task on randomly generated long texts containing an embedded passkey to evaluate retrieval accuracy under long-context scenarios. Our experiments are conducted using several widely adopted long-context language models, including Llama-3.1-8B-Instruct [18], Llama-2-7b-chat-hf [19], and Llama-2-7B-32K [20].

Baselines. Since the Quest algorithm does not reduce the overall GPU HBM footprint, but rather lowers computational overhead by selectively loading and computing a portion of the KV cache, it is unable to complete decoding in memory-constrained scenarios. Therefore, we have implemented several classic cache replacement algorithms as baselines for situations where GPU HBM is limited. Specifically, we use the following baselines for comparison in our experiments:

- Quest: The original Quest algorithm stores all KV cache data and the computed metadata entirely in the GPU. During each decoding step, it calculates attention scores using the Q vector of the current decoding token and the metadata. It then selects the top- k pages from the complete KV cache to perform the attention calculation.
- Quest with First Fit (FF): In scenarios with limited GPU HBM, we allocate a fixed-size buffer within the GPU memory to serve as the KV cache buffer. When a cache miss occurs during decoding, i.e., a required page is not resident in the GPU buffer, an eviction policy is triggered. FF addresses this by traversing the cache and evicting the first page it encounters that was not accessed in the current decoding step.
- Quest with LRU: With the LRU policy, we maintain an ordered list of indices of all cached pages. This list is

sorted based on the lexicographical order of the page access times. That is, when a batch of pages is accessed simultaneously, their positions at the head of the list are determined by their original relative order prior to the access. When a cache miss triggers an eviction, the last page in the list will be evicted.

- **Quest with Least Frequently Used (LFU):** In the LFU approach, an access frequency counter is maintained for each cached page and incremented upon every access during decoding. When a cache miss triggers an eviction, the page with the lowest access count among all resident pages is identified and selected for eviction.

B. Overall Performance

Cache Hit Ratio and Decoding Latency. To investigate the impact of LiteKV on LLM inference under GPU HBM constraints, we evaluated both the cache hit ratio and decoding latency. These iterations were conducted under the cache size of 64 pages. We used the Llama-2-7b-chat-hf model for our experiments. For the cache hit ratio evaluation, we initiated a prefill phase with 5,000 tokens, followed by 5,000 decoding iterations. As depicted in Fig. 4, LiteKV demonstrates a significant improvement in cache hit ratio, achieving at least a 17.76% increase compared to baseline methods. Even without accounting for relaxed hits (hit alternative page within the ϵ tolerance of attention score), LiteKV’s performance remains comparable to the leading baseline. This enhanced cache hit ratio directly contributes to lower decoding latency and reduced PCIe bandwidth consumption in practical applications.

Regarding decoding latency, our experiments were performed under GPU HBM constraints, where the primary overhead during word decoding is attributed to the transfer latency of pages not present in the current decoding process. We measured the average decoding latency using text from the GovReport [21] dataset in LongBench, with results presented in Fig. 4. Compared with Quest, LiteKV reduces the single decoding latency by at least 54.84% under these GPU HBM constrained conditions. While LiteKV significantly improves the cache hit ratio, the corresponding reduction in decoding latency, though substantial, does not exhibit a strictly linear relationship with the increase in cache hits, due to the startup overhead inherent in page transmission.

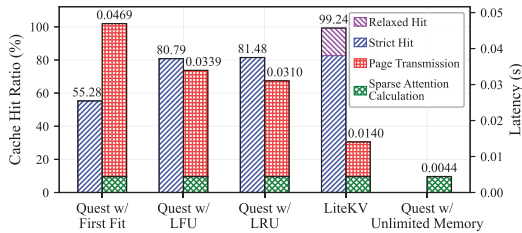


Fig. 4. Comparison of cache hit ratio and decoding latency.

Memory Reduction. We conducted memory reduction experiments on the LLaMA-2-7B-chat-hf model, measuring the impact of different memory reduction ratios on the decoding

latency of LiteKV. We also compared its performance against LRU, LFU, and First Fit algorithms. The results are presented in Fig. 5. Results show that even with memory reduced by up to 75%, LiteKV maintains high decoding efficiency compared to LRU, demonstrating its superior inference performance in memory-constrained edge scenarios.

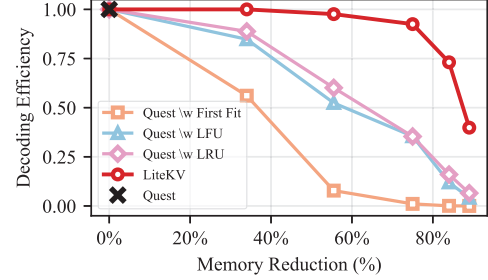


Fig. 5. Trade-off between decoding efficiency and memory.

Quality of Generated Text Under LiteKV. We evaluate the quality of text generated by models using LiteKV on two key benchmarks: the passkey retrieval task and perplexity on the PG19 dataset. The passkey retrieval task is designed to test a model’s ability to recall specific information from a long context, where a simple ‘passkey’ is hidden deep within the text. Perplexity (PPL) is a standard metric for language modeling that measures how well a model predicts a given text, where a lower perplexity score indicates higher fluency and coherence. We use the LLaMA-2-7B-32K model to perform inference on the first 2,000 tokens of the PG19 dataset and compute the token-level perplexity for each token. As shown in Fig. 6, the results indicate that for smaller values of ϵ , e.g., $\epsilon = 0.05$ or $\epsilon = 0.1$, LiteKV even outperforms Quest, which uses strict top- k pages without relaxation.

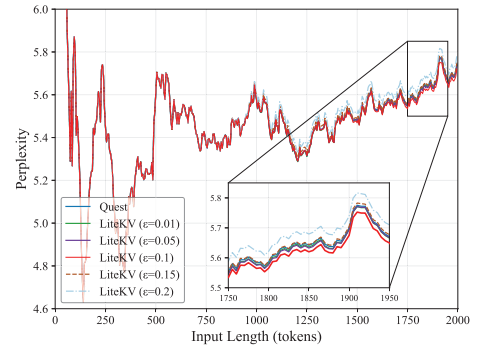


Fig. 6. Perplexity comparison.

Consistently high text generation quality was also observed in the passkey retrieval task. We here used the LLaMA-3.1-8B-Instruct model randomly embedding passkeys at varying depths within 10K-token sequences, and tasked the model with identifying them. As shown in Table I, under a smaller token budget, an appropriate selected value of ϵ can even outperform Quest, which utilizes the full top- k token set.

TABLE I
RESULTS OF 10K LENGTH PASSKEY RETRIEVAL TEST ON
LLAMA-3.1-8B-INSTRUCT

Method/Budget	16	32	64	128
Quest	80%	81%	94%	100%
LiteKV ($\epsilon = 0.05$)	80%	83%	95%	100%
LiteKV ($\epsilon = 0.1$)	76%	76%	89%	100%
LiteKV ($\epsilon = 0.2$)	66%	70%	86%	99%
LiteKV ($\epsilon = 0.3$)	65%	69%	84%	100%

C. Sensitivity Study

Impact of Parameter ϵ . The parameter ϵ plays a crucial role in LiteKV, as it directly influences the handling of KV cache pages that are not resident in the GPU HBM cache during the current decoding iteration. Intuitively, a larger ϵ allows more non-top- k pages to substitute top- k pages, whereas a smaller ϵ imposes stricter selection criteria, resulting in more frequent page replacement and corresponding I/O transmission. The impact of ϵ on the cache hit ratio is illustrated in Fig. 7. We conducted experiments under three different cache size settings, and the results consistently show that increasing ϵ leads to a higher cache hit ratio. Notably, we observe an inflection point in the trend, for instance, at $\epsilon = 0.05$ when the cache size is 32 pages.

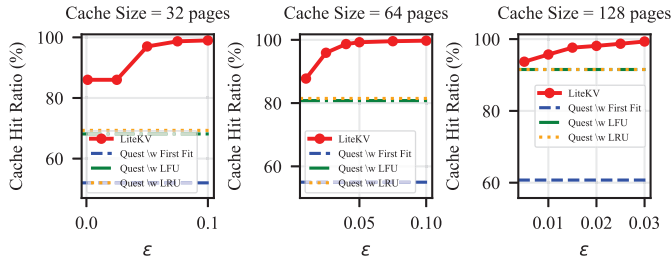


Fig. 7. Impact of ϵ on cache hit ratio under different cache sizes.

We further investigated the impact of the ϵ parameter on text generation quality, using perplexity as the evaluation metric. As shown in Fig. 8, perplexity follows a consistent trend across different cache sizes, initially decreasing with increasing ϵ , and then rising beyond a certain threshold. With a well-chosen ϵ , the perplexity can even outperform that of the original Quest. Combined with the previous experiments, this result suggests that for a fixed cache size, selecting an appropriate ϵ can effectively balance decoding efficiency and generation quality.

Impact of Decoding Context Length. As the decoding context length increases, the most evident change is a gradual increase in the total number of KV cache pages, which in turn directly impacts the cache hit ratio for the top- k pages required in GPU HBM during decoding. To examine this effect, we analyzed the impact of context length on the decoding latency of three different algorithms. The results are shown in Fig. 9. It is worth noting that Quest was evaluated

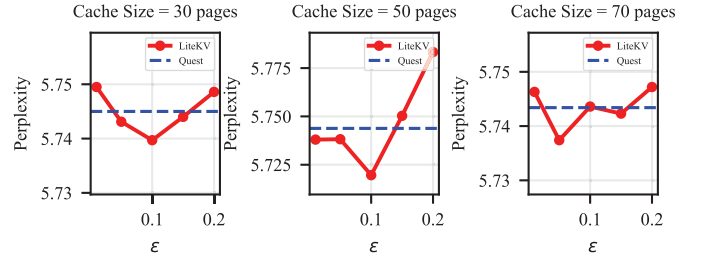


Fig. 8. Impact of ϵ on perplexity under different cache sizes.

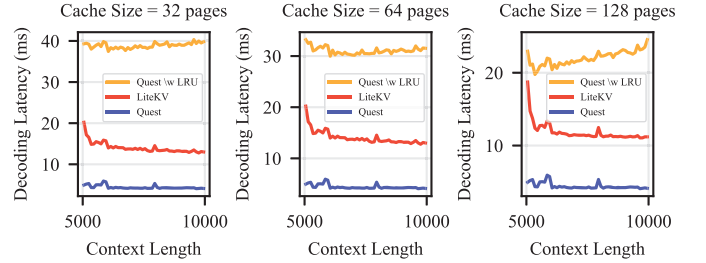


Fig. 9. Impact of context length on decoding latency.

under unconstrained memory conditions, thereby achieving optimal efficiency across all cache size settings. The results indicate that, across all cache sizes, the decoding latency of LiteKV remains unaffected by context length. In contrast, Quest with LRU exhibits increased latency as the context length grows, particularly when the cache size is set to 128 pages. These findings highlight the robustness and scalability of LiteKV under varying context lengths.

VI. RELATED WORKS

As a key component of LLMs, optimizing the process of attention calculation has become a primary approach to improving inference efficiency and reducing memory usage, which is especially critical in scenarios involving long contexts and limited resources. Existing efforts can be categorized into three lines of work: query-agnostic sparse attention, query-aware attention, and other memory reduction techniques such as quantization and compression. These works explore the trade-off from different perspectives among computational cost, memory savings, and generation quality.

A. Query-Agnostic Sparse Attention

Compared with full attention, sparse attention retains only the KV pairs of tokens that contribute most significantly during token generation, reducing computational and memory overhead in long-context scenarios. Recent studies have explored sparse attention strategies to suit diverse scenarios. Zhang et al. introduced H2O [1], which identifies heavy hitters, a small subset of tokens contributing most to attention, and retains both recent and high-impact tokens. Xiao et al. [2] leverage the phenomenon of attention sinks, where initial tokens consistently attract high attention, to construct a hybrid cache strategy StreamingLLM that ensures stable performance. Li

et al. [3] proposed SnapKV, which compresses the KV cache based on a clustering algorithm, retaining only the key features. Rather than compressing only the generated KV cache, Chen et al. [22] introduce a layer-wise pyramidal scheme PyramidInfer, that reduces memory usage during both the prefill and decoding phases by exploiting context redundancy and recent attention consistency across transformer layers. However, these methods rely on heuristics to aggressively discard past KVs, which means once a KV page is evicted, it cannot participate in the subsequent generation process, inevitably impacting the quality of generation.

B. Query-Aware Sparse Attention

To address quality degradation with long-context generation, some works try to estimate the importance of past tokens with respect to the current query to guide attention computation, such as SparQ [5], Double Sparsity [6], Quest [4], LServe [7], and Twilight [8]. Generally, these methods are also categorized as sparse attention in other literature, since only a small subset of tokens that are most valuable to the current query will participate in the attention computation. However, unlike the aforementioned KV eviction schemes, these works will typically retain the full KVs for all tokens in memory, ensuring that no potentially valuable tokens are omitted during generation. While these methods ensure high token generation quality while maintaining decoding efficiency, this design makes them less suitable for memory-limited edge devices, where maintaining the entire KV history is impractical.

C. Other Memory Reduction Techniques

Beyond sparse attention, a complementary direction explores compression, quantization, and hardware-optimized techniques to reduce the memory footprint of the KV cache. A series of works represented by KVQuant [23], Dynamic Memory Compression [24], GEAR [25], FlashAttention [26], AdaSplash [27], T-MAC [28], TensAllo [29], QLLMS [30], and NSA [31] propose various ways to quantize pages of KV matrices or compress memory layouts to improve storage efficiency. While these methods are broadly applicable and can yield significant memory savings, they generally operate at a coarse granularity and do not exploit the query-dependent importance of individual KV pages. As such, they may miss opportunities for finer-grained memory reuse.

In this work, we propose a cache-efficient KV replacement algorithm LiteKV, which is query-aware while only loading the most important KV pages into GPU memory, enabling efficient decoding under strict memory constraints. Unlike query-aware methods that retain full KV contexts in GPU memory, our approach can significantly improve cache utilization, resulting in a higher decoding efficiency. Moreover, LiteKV is orthogonal to the above related works, which means LiteKV can work with a well-established sparse attention mechanism or quantization techniques.

VII. DISCUSSION

Beyond Top- k Sparse Attention. This work primarily focuses on cache replacement under strict top- k sparse attention. Both

our theoretical analysis and experimental results demonstrate that allowing a modest relaxation in token selection can significantly improve cache hit ratio, thereby enhancing decoding efficiency in memory-constrained inference. This insight is not confined to top- k sparse attention alone. Similar principles can be extended to other sparse attention strategies, such as top- p , to facilitate their deployment in edge environments. However, unlike top- k , the theoretical analysis of top- p cache replacement is substantially more challenging. Even with distinct scores, finding the optimal token subset for top- p attention can be proven NP-hard through a reduction from the set cover problem, making it difficult to derive provable performance guarantees. Extending our framework to support top- p selection and analyzing its theoretical bounds remains an important direction for future work.

Learning-Based KV Cache Replacement. In LiteKV, cache replacement is driven by an LRU-like strategy that relies on historical token access patterns. Yet in LLMs, each token is generated based on previous tokens, which suggests that future attention scores may be predictable. Consequently, learning-based cache replacement methods, such as those that leverage page-level predictors [32], [33], hold promise for improving cache efficiency. These approaches could potentially surpass the theoretical limits of purely heuristic strategies while still maintaining formal performance guarantees. However, practical deployment of such methods poses new challenges. Given the high speed and low latency expectations of LLM inference, especially at the edge, the additional computational overhead introduced by predictors must be carefully balanced.

VIII. CONCLUSION

This paper addresses the overlooked challenge of KV cache replacement in memory-constrained edge environments for LLM inference. We analyze the theoretical complexity of the caching problem with top- k sparse attention and introduce LiteKV, a lightweight yet effective online algorithm that leverages $(1 + \epsilon)$ -relaxation to trade slight attention precision for substantial gains in cache efficiency. The competitive ratio of LiteKV is also given. Empirical results based on real-world deployment validate its effectiveness across diverse models and benchmarks, showing consistent improvements in both memory usage and decoding performance. Our study highlights the potential of relaxed token selection strategies for practical edge deployment and opens pathways for integrating learning-augmented cache policies in future works.

ACKNOWLEDGEMENT

This work is partially supported by the 2030 National Key AI Program of China under Grant 2021ZD0110400, NSFC under Grant 62132009, the Fundamental Research Funds for the Central Universities at China, Horizon Europe CODECO project (Grant No. 101092696), and the Alexander von Humboldt Foundation. Haisheng Tan and Li Zhang are the corresponding authors.

REFERENCES

- [1] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett *et al.*, “H₂O: Heavy-hitter oracle for efficient generative inference of large language models,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 34 661–34 710, 2023.
- [2] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, “Efficient streaming language models with attention sinks,” in *ICLR 2024*.
- [3] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen, “SnapKV: LLM knows what you are looking for before generation,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 22 947–22 970, 2024.
- [4] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han, “Quest: Query-aware sparsity for efficient long-context LLM inference,” *arXiv preprint arXiv:2406.10774*, 2024.
- [5] L. Ribar, I. Chelombiev, L. Hudlass-Galley, C. Blake, C. Luschi, and D. Orr, “SparQ Attention: Bandwidth-efficient LLM inference,” in *ICML 2024*.
- [6] S. Yang, Y. Sheng, J. E. Gonzalez, I. Stoica, and L. Zheng, “Post-training sparse attention with double sparsity,” *arXiv preprint arXiv:2408.07092*, 2024.
- [7] S. Yang, J. Guo, H. Tang, Q. Hu, G. Xiao, J. Tang, Y. Lin, Z. Liu, Y. Lu, and S. Han, “LServe: Efficient long-sequence LLM serving with unified sparse attention,” in *MLSys 2025*.
- [8] C. Lin, J. Tang, S. Yang, H. Wang, T. Tang, B. Tian, I. Stoica, S. Han, and M. Gao, “Twilight: Adaptive attention sparsity with hierarchical top-*p* pruning,” *arXiv preprint arXiv:2502.02770*, 2025.
- [9] L. A. Belady, “A study of replacement algorithms for a virtual-storage computer,” *IBM Systems journal*, vol. 5, no. 2, pp. 78–101, 1966.
- [10] P. J. Denning, “The working set model for program behavior,” *Communications of the ACM*, vol. 11, no. 5, pp. 323–333, 1968.
- [11] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, “Atom: Low-bit quantization for efficient and accurate LLM serving,” *Proceedings of Machine Learning and Systems*, vol. 6, pp. 196–209, 2024.
- [12] R. M. Karp, “Reducibility among combinatorial problems,” in *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*. Springer, 2009, pp. 219–241.
- [13] M.-K. Lee, P. Michaud, J. S. Sim, and D. Nyang, “A simple proof of optimality for the MIN cache replacement policy,” *Information Processing Letters*, vol. 116, no. 2, pp. 168–170, 2016.
- [14] D. D. Sleator and R. E. Tarjan, “Amortized efficiency of list update and paging rules,” *Communications of the ACM*, vol. 28, no. 2, pp. 202–208, 1985.
- [15] J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap, “Compressive transformers for long-range sequence modelling,” *arXiv preprint arXiv:1911.05507*, 2019.
- [16] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou *et al.*, “LongBench: A bilingual, multitask benchmark for long context understanding,” *arXiv preprint arXiv:2308.14508*, 2023.
- [17] B. Peng, J. Quesnelle, H. Fan, and E. Shippole, “Yarn: Efficient context window extension of large language models,” *arXiv preprint arXiv:2309.00071*, 2023.
- [18] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, “The Llama 3 herd of models,” *arXiv e-prints*, pp. arXiv–2407, 2024.
- [19] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, “Llama 2: Open foundation and fine-tuned chat models,” *arXiv preprint arXiv:2307.09288*, 2023.
- [20] T. Computer, “LLaMA-2-7B-32K,” <https://huggingface.co/togethercomputer/LLaMA-2-7B-32K>, 2023.
- [21] L. Huang, S. Cao, N. Parulian, H. Ji, and L. Wang, “Efficient attentions for long document summarization,” *arXiv preprint arXiv:2104.02112*, 2021.
- [22] D. Yang, X. Han, Y. Gao, Y. Hu, S. Zhang, and H. Zhao, “PyramidInfer: Pyramid kv cache compression for high-throughput LLM inference,” in *ACL 2024*.
- [23] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami, “KVQuant: Towards 10 million context length LLM inference with kv cache quantization,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 1270–1303, 2024.
- [24] P. Nawrot, A. Łańcucki, M. Chochowski, D. Tarjan, and E. Ponti, “Dynamic memory compression: Retrofitting LLMs for accelerated inference,” in *ICML 2024*.
- [25] H. Kang, Q. Zhang, S. Kundu, G. Jeong, Z. Liu, T. Krishna, and T. Zhao, “Gear: An efficient kv cache compression recipe for near-lossless generative inference of LLM,” *arXiv preprint arXiv:2403.05527*, 2024.
- [26] J. Shah, G. Bikshandi, Y. Zhang, V. Thakkar, P. Ramani, and T. Dao, “FlashAttention-3: Fast and accurate attention with asynchrony and low-precision,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 68 658–68 685, 2024.
- [27] N. Gonçalves, M. V. Treviso, and A. Martins, “AdaSplash: Adaptive sparse flash attention,” in *ICML 2025*.
- [28] J. Wei, S. Cao, T. Cao, L. Ma, L. Wang, Y. Zhang, and M. Yang, “T-mac: CPU renaissance via table lookup for low-bit LLM deployment on edge,” in *EuroSys 2025*.
- [29] B. Zhang, J. Zhang, J. Hou, and Y. Wang, “Tensallo: Adaptive deployment of LLMs on resource-constrained heterogeneous edge devices,” in *IEEE INFOCOM 2025*.
- [30] M. Hu, Q. He, and D. Wu, “QLLMS: Quantization-adaptive LLM scheduling for partially informed edge serving systems,” in *IEEE INFOCOM 2025*.
- [31] J. Yuan, H. Gao, D. Dai, J. Luo, L. Zhao, Z. Zhang, Z. Xie, Y. Wei, L. Wang, Z. Xiao *et al.*, “Native sparse attention: Hardware-aligned and natively trainable sparse attention,” *arXiv preprint arXiv:2502.11089*, 2025.
- [32] T. Lykouris and S. Vassilvitskii, “Competitive caching with machine learned advice,” *Journal of the ACM (JACM)*, vol. 68, no. 4, pp. 1–25, 2021.
- [33] M. Elias, H. Kaplan, Y. Mansour, and S. Moran, “Learning-augmented algorithms with explicit predictors,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 97 972–98 008, 2024.