EFFICIENT LONG-CONTEXT PROCESSING WITH KV CACHE: METHODS, CHALLENGES, AND FUTURE DIRECTIONS

by

XIANG LIU

Data Science and Analytics Thrust, Information Hub
The Hong Kong University of Science and Technology (Guangzhou)
Supervised by Prof. Xiaowen CHU & Prof. Xuming HU

March 2025, Guangzhou

EFFICIENT LONG-CONTEXT PROCESSING WITH KV CACHE: METHODS, CHALLENGES, AND FUTURE DIRECTIONS

by

XIANG LIU

Data Science and Analytics Thrust, Information Hub

The Hong Kong University of Science and Technology (Guangzhou)

ABSTRACT

Large Language Models (LLMs) have demonstrated remarkable capabilities across various tasks, but processing long text sequences presents significant computational and memory challenges. The Key-Value (KV) Cache, which stores intermediate attention computations to avoid redundant calculations during autoregressive generation, is a critical component where optimization can substantially improve efficiency. This thesis presents a comprehensive survey of KV Cache optimization techniques and introduces novel methods for enhancing long-context processing in LLMs.

We begin by analyzing the fundamental challenges in KV Cache management, establishing a formal problem definition that highlights the linear growth of memory requirements with sequence length and the quadratic computational complexity of attention mechanisms. We then propose a systematic taxonomy of KV Cache optimization approaches, categorizing them into KV Cache Selection (static and dynamic methods), KV Cache Budget Allocation (layer-wise and head-wise strategies), and examining their theoretical foundations and practical implementations.

Our research makes three significant contributions to the field. First, we introduce LongGenBench, a novel benchmark specifically designed to evaluate long-context generation capabilities beyond mere information retrieval. Second, we propose ChunkKV, a

semantic-aware compression framework that preserves contextual integrity by treating semantically coherent token groups as fundamental compression units. Third, through extensive experimental analysis, we provide a comprehensive evaluation of how different compression techniques affect various model capabilities, revealing task-dependent sensitivity patterns and demonstrating that arithmetic reasoning tasks are particularly vulnerable to compression artifacts.

Our findings suggest that while significant progress has been made in improving long-context processing efficiency, future innovations should focus on task-adaptive compression strategies, compression-aware training techniques, and theoretical frameworks for understanding performance bounds under different optimization constraints.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Large language models (LLMs) have demonstrated remarkable capabilities in processing and generating text across various applications [45, 46, 56], from document understanding [39, 41] to code generation [27], multi-turn dialogues [12, 68], complex reasoning tasks [26, 22], and essay writing [49, 28]. The demand for enhanced long-context capabilities in LLMs has grown exponentially as applications increasingly require processing extensive documents, maintaining coherent multi-turn conversations, and performing complex reasoning across thousands or even millions of tokens. This long-context understanding enables models to capture distant dependencies, maintain consistency throughout lengthy generations, and extract relevant information from substantial bodies of text. Furthermore, recent advances have pushed context windows from several thousand tokens to millions [55, 53], dramatically expanding the potential use cases for LLMs while simultaneously intensifying the computational and memory challenges associated with these extended sequence lengths.

The Decoder-Only attention mechanism employed in these LLMs poses a significant challenge for handling long-context sequences. As these models grow in size and complexity, the computational complexity increases quadratically with context length. During inference, the KV cache stores key and value matrices for the Decoder-Only attention mechanism, which reduces the time complexity when generating new tokens. Nevertheless, maintaining the KV cache requires substantial GPU memory, with usage scaling linearly with the number of layers, attention heads, head dimension, and sequence length. This memory constraint becomes particularly problematic for LLMs exceeding 100B parameters, such as GPT-4 [18], DeepSeek-R1 [22], and LLaMA-3.1-405B [15]. Furthermore, in LLM serving platforms, service providers must maintain separate KV caches for each user conversation, presenting a considerable resource challenge. In response, KV Cache-centric LLM serving platforms [44] have been proposed to address these scaling difficulties and optimize memory usage during inference.

This survey offers a comprehensive examination of both traditional and emerging approaches to KV cache compression in large language models (LLMs), presenting a clear developmental trajectory for current methodologies. First, we explore how LLMs have

demonstrated remarkable capabilities in processing and generating text across various long-context scenarios, including retrieval, summary, reasoning, and generation. Our benchmark, LongGenBench [35], is the first comprehensive benchmark for long-context generation, efficient and effective for evaluating the performance when LLMs are under the long response requirement. Then we review the state-of-the-art KV cache compression techniques, including statical eviction, layer-wise dynamic eviction, and semantic-aware chunk-wise eviction techniques [37]. We also identify the KV Cache compression are still facing the performance gap on the real world benchmark except long-context scenarios. Finally, we propose several promising future research directions with significant implications for both academic advancement and industrial deployment of large-scale language models.

The structure of this survey is organized as follows: Chapter 2 introduces the foundations of Decoder-Only attention mechanisms and explains how their computational complexity increases quadratically with context length and preliminary for KV Cache compression. Chapter 3 provides an extensive review of long-context benchmarks, examining how recent models perform when processing documents spanning thousands to millions of tokens. In Chapter 4, we present a comprehensive analysis of KV cache compression techniques, highlighting their effectiveness and limitations in various deployment scenarios. Chapter 5 presents our novel contributions to both long-context evaluation and KV cache optimization. Finally, Chapter 6 proposes several promising future research directions with significant implications for both academic advancement and industrial deployment of large-scale language models.

CHAPTER 2

PRELIMINARY

This chapter provides essential background information on transformer architectures, attention mechanisms, and the Key-Value (KV) Cache, establishing the technical foundation for our comprehensive analysis of long-context processing in Large Language Models (LLMs).

2.1 Transformer Architecture and Attention Mechanisms

2.1.1 Decoder-Only Architecture

Contemporary large-scale language models, including GPT-4 [18], DeepSeek-R1 [22], and LLaMA-3.1-405B [15], predominantly implement the Decoder-Only transformer architecture as illustrated in Figure 2.1. This architectural paradigm consists of a sequence of transformer blocks, wherein each block incorporates masked self-attention mechanisms and position-wise feed-forward neural networks. The inference process in such models adheres to an autoregressive formulation, whereby text generation proceeds in a strictly sequential manner—the probability distribution for each subsequent token x_t is conditionally dependent on the complete preceding sequence x_t. A salient characteristic of the Decoder-Only architecture lies in its computational asymmetry: the architecture facilitates parallelization during the training phase through masked attention, while necessitating sequential processing during inference due to the autoregressive constraint. Empirical evidence demonstrates that model performance exhibits monotonic improvement with increases in both parameter count and context length; however, this scaling paradigm introduces substantial computational and memory complexity that requires sophisticated optimization techniques to maintain practical deployability in resource-constrained environments.

2.1.2 Attention Mechanism Complexity Analysis

The attention mechanism, despite its effectiveness in modeling contextual relationships, introduces significant computational challenges in processing long sequences. In the standard transformer architecture, the self-attention operation for a given layer l can be formalized as follows:

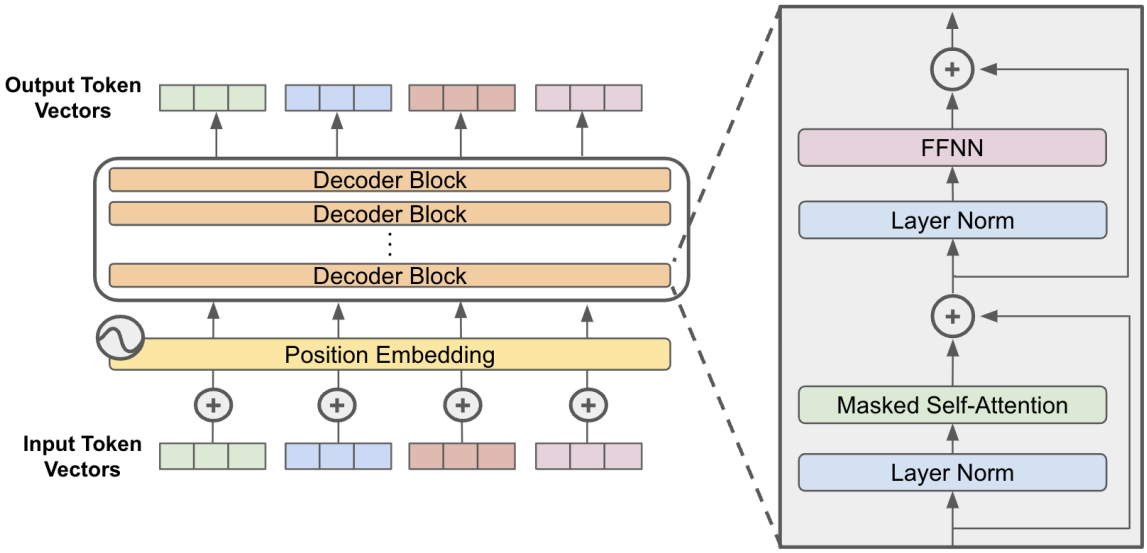


Figure 2.1. Decoder-Only Architecture [8]

Given an input representation $\mathbf{X} \in \mathbb{R}^{N \times D}$, where N represents the sequence length and D denotes the hidden dimension size, the multi-head attention mechanism first projects the input into queries, keys, and values through learnable parameters:

$$\mathbf{Q}^{l} = \mathbf{X}\mathbf{W}_{Q}^{l} \in \mathbb{R}^{N \times D} \tag{2.1}$$

$$\mathbf{K}^{l} = \mathbf{X}\mathbf{W}_{K}^{l} \in \mathbb{R}^{N \times D} \tag{2.2}$$

$$\mathbf{V}^{l} = \mathbf{X}\mathbf{W}_{V}^{l} \in \mathbb{R}^{N \times D} \tag{2.3}$$

where $\mathbf{W}_{Q}^{l}, \mathbf{W}_{K}^{l}, \mathbf{W}_{V}^{l} \in \mathbb{R}^{D \times D}$ are learnable parameter matrices. In multi-head attention with H heads, these projections are further projected and split into H distinct representations, each operating in a lower-dimensional space $D_h = D/H$:

$$\mathbf{Q}^{l,h} = \mathbf{X}\mathbf{W}_{Q}^{l,h} \in \mathbb{R}^{N \times D_h} \tag{2.4}$$

$$\mathbf{K}^{l,h} = \mathbf{X}\mathbf{W}_{K}^{l,h} \in \mathbb{R}^{N \times D_h} \tag{2.5}$$

$$\mathbf{V}^{l,h} = \mathbf{X}\mathbf{W}_{V}^{l,h} \in \mathbb{R}^{N \times D_h} \tag{2.6}$$

where $\mathbf{W}_{Q}^{l,h}, \mathbf{W}_{K}^{l,h}, \mathbf{W}_{V}^{l,h} \in \mathbb{R}^{D \times D_h}$ are head-specific projection matrices.

For each head h, the attention scores are computed as:

$$\mathbf{A}^{l,h} = \text{softmax}\left(\frac{\mathbf{Q}^{l,h}(\mathbf{K}^{l,h})^{T}}{\sqrt{D_h}}\right) \in \mathbb{R}^{N \times N} \tag{2.7}$$

The scaled dot-product attention output for each head is:

$$\mathbf{O}^{l,h} = \mathbf{A}^{l,h}\mathbf{V}^{l,h} \in \mathbb{R}^{N\times D_h} \tag{2.8}$$

Finally, the outputs from all heads are concatenated and linearly transformed:

$$\mathbf{O}^{l} = [\mathbf{O}^{l,1}, \mathbf{O}^{l,2}, \ldots, \mathbf{O}^{l,H}]\mathbf{W}_O^l \in \mathbb{R}^{N\times D} \tag{2.9}$$

where $\mathbf{W}_O^l \in \mathbb{R}^{D\times D}$ is a learnable parameter matrix.

Computationally, this process requires $O(N^2D)$ operations due to the matrix multiplication $\mathbf{Q}^{l,h}(\mathbf{K}^{l,h})^T$, and the memory footprint likewise scales quadratically with sequence length, as the attention weight matrix $\mathbf{A}^{l,h}$ occupies $O(N^2)$ storage space. This quadratic scaling property creates a fundamental bottleneck when extending context windows beyond certain thresholds, necessitating architectural innovations or efficiency optimizations to maintain practical deployability.

During inference, specifically in the autoregressive decoding phase that employs KV Cache, the previously computed keys $\mathbf{K}_t^{l,h}$ and values $\mathbf{V}_t^{l,h}$ for all positions up to time step t are stored and reused, which relates directly to the KV Cache formulation $\mathscr{C}_t = \{\mathbf{K}_t^{l,h}, \mathbf{V}_t^{l,h}\}_{l=1,h=1}^{L,H}$ presented in Section 2.3.

2.2 Long-context Processing: Evaluation and Cost Analysis

2.2.1 Long-context Evaluation Paradigms

The assessment of long-context understanding capabilities in transformer-based language models presents unique methodological challenges. Current evaluation frameworks can be categorized into several distinct paradigms, each with specific strengths and limitations.

Retrieval-based evaluations examine a model's ability to identify and extract information from distant parts of an extended document. Notable benchmarks include SCROLLS [48], which aggregates diverse tasks requiring long-range understanding, and LongBench [4, 5], which evaluates performance across multiple languages and domains. These assessments typically measure precision, recall, and F1 scores on information extraction tasks at varying context distances. Ruler [25] is a new benchmark for long-context evaluation, combining different Needle-in-Haystack strategies. Specialized needle-in-haystack tests explicitly measure positional recall by embedding critical information at various distances within

deliberately extended contexts. These controlled experiments reveal attention decay patterns and context utilization efficiency, with most models demonstrating significant performance degradation beyond specific positional thresholds.

Reasoning evaluations focus on the model's capacity to synthesize information across lengthy contexts to draw logical conclusions. Examples include L-Eval [3] and NarrativeQA [30], which require models to answer questions that demand integration of dispersed textual elements. Performance on such tasks correlates strongly with a model's effective context window, though with diminishing returns beyond certain thresholds. LongGenBench [35] is a new benchmark for long-context generation under arithmetic, world knowledge, and commonsense reasoning tasks.

2.2.2 Computational Costs

The extension of context windows in transformer models entails substantial economic and computational costs that scale non-linearly with context length. These costs manifest across multiple dimensions:

Computationally, the aforementioned $O(N^2D)$ complexity of attention operations translates to quadratic increases in FLOP requirements as context length extends. Empirical measurements indicate that doubling context length typically increases inference time by 3-4× when accounting for memory access patterns and hardware utilization efficiencies. For a 70B parameter model, extending context from 8K to 32K tokens can increase per-inference computation by approximately an order of magnitude.

From an economic perspective, the increased computational demands directly affect operational expenses. Cloud-based inference costs scale proportionally with computation time, resulting in significantly higher per-query expenses for long-context processing. Current estimates suggest that processing a 32K context query on a 70B model incurs 5-7× the operational cost of an equivalent 4K context query, primarily due to extended GPU reservation times and memory requirements.

Memory consumption poses perhaps the most immediate constraint, as the KV Cache memory requirement scales linearly with context length as established in our formulation $M(\mathscr{C}_t) = 2 \times L \times H \times D_h \times t \times s$. This scaling necessitates specialized hardware configurations or distributed inference strategies when handling extended contexts, further increasing deployment complexity and cost.

2.3 KV Cache: Problem Formalization

The Key-Value (KV) Cache is a critical optimization technique in transformer-based LLMs that significantly improves inference efficiency during autoregressive decoding. In this section, we establish a formal notation framework and precisely define the KV Cache optimization problem.

2.3.1 Notation and Definitions

An LLM fundamentally maps an input token sequence $X = (x_1, x_2, \ldots, x_n)$ to a series of probability distributions $P = (p_1, p_2, \ldots, p_n)$, where each $x_i \in \mathcal{V}$ represents a token from vocabulary $\mathcal{V}$ of size $|\mathcal{V}| = V$, and each $p_i \in \mathbb{R}^V$ represents a probability distribution over the next possible token. During autoregressive generation, the model typically samples from p_n to determine the next token.

The computational architecture of modern decoder-only LLMs comprises:

- An embedding layer that maps each token x_i to a D-dimensional vector $\mathbf{h}_i^{(0)}$
- L sequential transformer decoder blocks
- A final linear projection with softmax that transforms $\mathbf{h}_i^{(L)}$ into p_i

Each transformer decoder block contains a self-attention mechanism and a feed-forward network. For our analysis of KV Cache, we focus on the self-attention component, which can be formalized as follows:

2.3.2 Self-Attention and KV Cache Formation

Within each layer l of a transformer decoder, the self-attention mechanism projects each hidden state $\mathbf{h}_i^{(l)}$ into queries, keys, and values:

$$\mathbf{q}_i^{(l)} = \mathbf{h}_i^{(l)} \mathbf{W}_Q^{(l)} \tag{2.10}$$

$$\mathbf{k}_i^{(l)} = \mathbf{h}_i^{(l)} \mathbf{W}_K^{(l)} \tag{2.11}$$

$$\mathbf{v}_i^{(l)} = \mathbf{h}_i^{(l)} \mathbf{W}_V^{(l)} \tag{2.12}$$

where $\mathbf{W}_Q^{(l)}$, $\mathbf{W}_K^{(l)}$, and $\mathbf{W}_V^{(l)}$ are learnable parameter matrices. In multi-head attention with H heads, each with dimension $D_h = D/H$, these projections are further partitioned, yielding per-head representations $\mathbf{q}_i^{(l,h)}$, $\mathbf{k}_i^{(l,h)}$, and $\mathbf{v}_i^{(l,h)}$ for each head $h \in \{1, 2, \ldots, H\}$.

The self-attention output for a sequence can be computed as:

$$\mathbf{H}'^{(l)} = \text{softmax}\left(\text{mask}\left(\frac{\mathbf{Q}^{(l)} \cdot \mathbf{K}^{(l)^\top}}{\sqrt{D}}\right)\right) \cdot \mathbf{V}^{(l)} \cdot \mathbf{W}_O^{(l)} \tag{2.13}$$

In decoder-only architectures, the causal mask $\text{mask}(\cdot)$ ensures that each token i can only attend to tokens j where $j \leqslant i$, preserving the autoregressive property during generation. This causality has profound implications for inference optimization.

2.3.3 KV Cache Definition and Memory Requirements

The KV Cache leverages this causal property: since earlier tokens' computations are independent of later tokens, we can store and reuse the key and value matrices during autoregressive generation. Formally, the KV Cache at time step t can be defined as:

$$\mathscr{C}_t = \{\mathbf{K}_t^{(l,h)}, \mathbf{V}_t^{(l,h)}\}_{l=1,h=1}^{L,H} \tag{2.14}$$

where $\mathbf{K}_t^{(l,h)} \in \mathbb{R}^{t \times D_h}$ and $\mathbf{V}_t^{(l,h)} \in \mathbb{R}^{t \times D_h}$ represent the keys and values for all tokens up to position t in layer l and head h.

The memory consumption of the KV Cache scales linearly with sequence length:

$$M(\mathscr{C}_t) = 2 \times L \times H \times D_h \times t \times s \tag{2.15}$$

where s represents the size in bytes of each element (typically 2 bytes for FP16 or 4 bytes for FP32).

For long context processing, this memory requirement becomes prohibitive. For example, with $L = 32$, $H = 32$, $D_h = 128$, $t = 32{,}000$, and $s = 2$ bytes, the KV Cache would require approximately 8.6 GB of memory, which can exceed available GPU memory, especially when batch processing multiple sequences.

2.3.4 KV Cache Optimization Problem

The core challenge in KV Cache optimization can be formalized as finding a function f that transforms the original cache $\mathscr{C}_t$ into a compressed representation $\mathscr{C}'_t$:

$$\mathscr{C}'_t = f(\mathscr{C}_t) \tag{2.16}$$

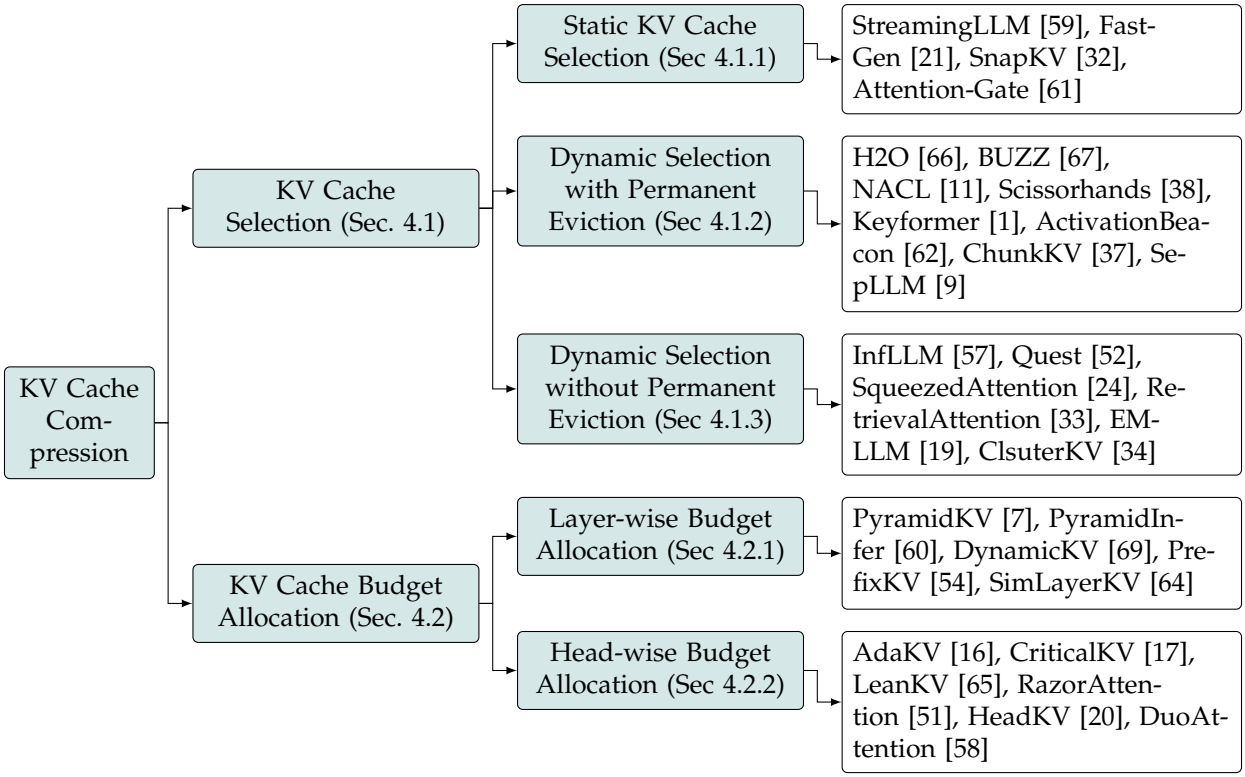
such that:

$$\text{Mem}(\mathscr{C}_t') \leqslant \text{Mem}(\mathscr{C}_t) \tag{2.17}$$

$$|\mathscr{LLM}_{\mathscr{C}_t'}(X) - \mathscr{LLM}_{\mathscr{C}_t}(X)| \leqslant \epsilon \tag{2.18}$$

where $\mathscr{LLM}_{\mathscr{C}}(X)$ represents the model output using cache $\mathscr{C}$, and ϵ is a tolerance threshold for output degradation. The optimization goal is to minimize memory requirements while maintaining model performance within acceptable bounds.

2.3.5 Taxonomy of KV Cache Compression Techniques



Based on our analysis of existing literature, we propose a comprehensive taxonomy of KV Cache optimization techniques categorized into three primary approaches as illustrated in Figure 2.3.5. First, *KV Cache Selection* approaches implement token filtering mechanisms that can be either static (predetermined patterns) or dynamic (content-dependent), with dynamic methods further differentiated by whether they permanently discard tokens (permanent eviction) or maintain them in secondary storage for potential reuse (without permanent eviction). Second, *KV Cache Budget Allocation* methods distribute limited memory resources non-uniformly, either prioritizing important attention heads (head-wise allocation) or allocating different context lengths to various layers (layer-wise allocation) based on their empirical importance to model performance. This taxonomy provides a structured framework for understanding the design space of KV Cache optimization techniques and guides our subsequent detailed analysis of each approach.

CHAPTER 3

LONG-CONTEXT EVALUATION

In this chapter, we introduce the commonly benchmark used in KV Cache optimization, followed by a presentation of the evaluation metrics.

3.1 Benchmark

3.1.1 Long-Context Benchmarks

Document-based Evaluation Methodologies Comprehensive assessment of LLMs' long-context capabilities frequently employs document-centric benchmarks requiring information extraction from extensive texts. These evaluation frameworks, including **L-Eval** [3], **ZeroSCROLLS** [47], **BAMBOO** [14], and **LongBench** [4, 5], present models with reference materials spanning 4,000-20,000 tokens, with extreme cases in **InfiniteBench** [63], **XL^{2}bench**[43], and **LooGLE** [31] approaching 200,000 tokens. Despite their ecological validity, these approaches present methodological challenges: they often conflate evaluation with the model's prior knowledge, employ fixed context lengths that resist parametric manipulation, and introduce assessment variability through uncontrolled generation. For targeted evaluation of long-sequence generation specifically, **LongGenbench** [35] provides a specialized framework addressing these limitations.

Controlled Retrieval Paradigms Alternative evaluation strategies employ controlled information retrieval tasks that isolate context processing capabilities from prior knowledge effects. These approaches, exemplified by **RULER** [25], **Needle in a Haystack** [29], and **Passkey Retrieval** [42], embed specific information within irrelevant context of adjustable length, creating precisely calibrated difficulty gradients. Such methodologies offer superior experimental control, implementation flexibility, and independence from model-specific knowledge bases. Figure 3.1 shows the visualization of the Needle in a Haystack benchmark. The X-axis represents the length of the context, and the Y-axis represents the depth of the needle, the color for each block represents the probability of the model to retrieve the needle. The primary critique of these approaches centers on their potential oversimplification of real-world context processing demands, potentially favoring shallow pattern recognition over deeper contextual understanding.

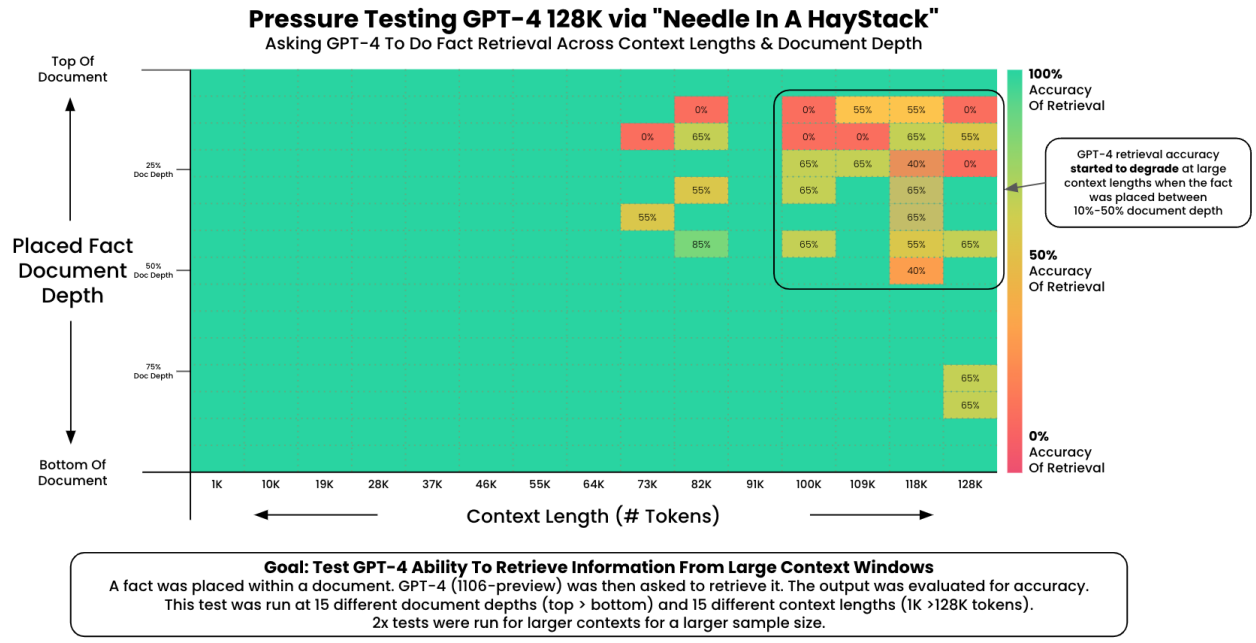


Figure 3.1. Needle in a Haystack Benchmark Visualization [29]

Synthetic Context Extension Beyond dedicated long-context evaluation frameworks, researchers also employ synthetic context extension techniques to assess model capabilities. By applying few-shot prompting [6, 36] or simulating extended conversation histories, traditional shorter benchmarks can be transformed into long-context assessment tools. This approach offers particular value for evaluating KV Cache optimizations under conditions that more accurately reflect typical deployment patterns, where most LLM interactions involve sequences of moderately-sized exchanges rather than single extensive documents. Such methodologies help quantify the relationship between context length, computational efficiency, and performance across varying interaction patterns.

3.1.2 Evaluation Metric

Generation Quality To comprehensively evaluate the impact of KV Cache compression techniques on model performance, we assess output quality across diverse benchmark categories including world knowledge, commonsense reasoning, arithmetic reasoning, code generation, and safety. For each benchmark, we compare the model's generation capabilities with and without compression using established metrics such as accuracy, F1 score, BLEU, and RougeL. These metrics provide quantitative measures of how closely the compressed model's outputs align with reference answers or the uncompressed model's responses. For each task category and compression method C, we calculate the relative performance change as follows:

$$\Delta P = \frac{P_C - P_{base}}{P_{base}} \tag{3.1}$$

where P_C and P_{base} represent the performance scores with and without compression, respectively. This normalized metric allows for direct comparison across different compression techniques and facilitates identification of methods that maintain generation quality while achieving significant memory reduction. Ideally, an effective compression approach should yield ΔP values close to zero, indicating minimal degradation in output quality despite substantial improvements in memory efficiency. Particularly sensitive benchmarks like arithmetic reasoning and code generation often reveal compression artifacts that might remain undetected in more general language tasks, making them especially valuable for comprehensive evaluation.

Perplexity The Perplexity (PPL) is: for each token, the model calculates the natural logarithm likelihood value of the probability distribution predicted based on its previous tokens, takes the average, and then calculates the value as the exponent of e, mathematically given by Formula 3.2, where ANLL refers to the average natural logarithm likelihood.

$$\text{ANLL} = -\frac{1}{N}\sum_{i=1}^{N} \log P\left(x_i | x_1 x_2 \ldots x_{i-1}\right), \quad \text{PPL} = e^{\text{ANLL}} \tag{3.2}$$

PPL can provide a rough reference for the performance changes of the model. If PPL rises sharply, it usually means that the model's ability has significantly decreased, such as completely losing language ability, etc.

Per Token GPU-Memory Usage For KV Cache, the most intuitive optimization indicator would be the memory space occupied by each token. The LLaMA2-7B model, as a typical example, theoretically occupies 0.5MB of memory for each KV Cache entry. Please note that when measuring this indicator rigorously, the fragment space generated by the token should also be taken into account, that is, it is best to measure according to the actual memory occupancy, rather than a value calculated through a string of parameters.

Throughput and Latency Throughput and latency are important indicators for measuring the time efficiency of a model. Throughput, usually measured in tokens per second (token/s), represents how many new tokens the model can generate per second. The higher the throughput, the better the model efficiency. In the Decoding phase, latency is usually considered to be the time required to generate each new token, which is the reciprocal of the throughput, typically in milliseconds.

CHAPTER 4

KV CACHE COMPRESSION TECHNIQUES

This chapter presents a comprehensive analysis of KV cache compression techniques for enhancing the efficiency of long-context processing in Large Language Models. We organize these techniques into three main categories: KV Cache Selection and KV Cache Budget Allocation. Each approach offers distinct trade-offs between memory efficiency, computational overhead, and output quality. The comprehensive taxonomy of KV Cache compression techniques is shown in Figure 2.3.5.

These methods are mostly training-free, which can be applied to any LLM model during the inference phase without introducing additional inference latency, but can significantly reduce memory consumption and improve inference throughput, especially in the case of long-context processing.

4.1 KV Cache Selection

KV Cache Selection techniques focus on identifying and retaining only the most important tokens in the context window, effectively reducing memory requirements by eliminating or deprioritizing less relevant information. These approaches can be categorized into static selection patterns and dynamic content-aware methods.

4.1.1 Static KV Cache Selection

Static selection techniques employ predetermined patterns for token retention, offering implementation simplicity and predictable performance characteristics. These methods typically involve fixed stride or positional importance assumptions. This research direction's pioneering methods discovered that KV Cache exhibits characteristics of high sparsity and specific patterns, providing both theoretical and experimental foundations for KV Cache compression.

StreamingLLM [59] pioneered the identification of the attention sink phenomenon in transformer architectures. This research revealed that retaining key-value pairs from the earliest tokens in a sequence is essential for maintaining model performance. The researchers observed that attention weights accumulate disproportionately at initial positions—a pattern that occurs independently of the semantic relevance of those tokens. By

strategically combining these attention sink positions with recent contextual information, StreamingLLM achieves efficient processing while preserving inference quality.

FastGen [21] presents an adaptive framework that analyzes attention patterns to optimize KV cache management. During the prompt processing stage, FastGen examines multiple attention structures and determines the optimal retention policy for individual attention heads. These policies are then applied during generation to selectively maintain the most valuable tokens. The approach recognizes diverse attention behaviors—including locality-sensitive patterns, punctuation-centric focus, sparsely distributed attention, and global attention spans—and applies tailored strategies for each. This pattern-recognition methodology enables FastGen to intelligently preserve tokens based on their functional roles (such as special tokens or punctuation), temporal proximity, or attention distribution characteristics, resulting in efficient memory utilization without compromising generation quality.

SnapKV [32] introduces a selective token retention mechanism based on information significance analysis. The research identifies that generation quality depends primarily on a subset of contextual elements rather than the entire prompt history. By implementing an end-positioned monitoring framework to evaluate token contributions, SnapKV effectively identifies and preserves the most influential components of the context. This approach allows the system to maintain performance while substantially reducing memory requirements by combining strategically identified high-value historical tokens with recent context from the monitoring region. The technique's effectiveness demonstrates that contextual relevance persists consistently throughout the generation process.

Attention-Gate [61] proposes an adaptive memory management framework that employs machine learning to optimize context retention. Unlike fixed-pattern approaches, this method implements a trainable neural component that evaluates the entire input sequence and determines which tokens should be preserved or discarded. By incorporating this parameterized decision-making system, Attention-Gate achieves contextually-aware memory optimization that can adjust to different content types and computational constraints. This learnable approach represents a shift from heuristic methods toward more flexible, content-responsive context management.

4.1.2 Dynamic Selection with Permanent Eviction

These approaches adaptively identify less important tokens based on context-specific criteria and permanently remove them from the KV cache, achieving substantial memory savings at the potential cost of information loss.

H2O [66] introduces an attention-driven approach based on the identification of "Heavy Hitters"—tokens that exert disproportionate influence on model computations. This method reconceptualizes the cache retention challenge through a dynamic submodular optimization framework. By analyzing the distribution and accumulation of normalized attention weights across the sequence, H2O identifies tokens with exceptional computational significance. The approach implements a dual retention strategy that preserves both these highly influential historical tokens and the most recent context window, recognizing that temporal proximity often correlates with contextual relevance. This balanced methodology enables efficient memory utilization while maintaining generation quality by focusing computational resources on genuinely impactful tokens.

BUZZ [67] implements an innovative context management strategy through segment-based distribution analysis. This approach divides the attention space into uniform regions and identifies representative tokens from each segment, resulting in multiple efficiency benefits. The method achieves linear computational complexity while leveraging parallel processing capabilities to enhance performance at scale. BUZZ's distinctive contribution lies in its biomimetic retention pattern that distributes tokens non-uniformly across the temporal spectrum—applying greater density to historical information and more selective retention to recent content. This graduated sampling methodology preserves both individual high-value tokens and the holistic attention distribution patterns, maintaining the structural integrity of the context while optimizing memory utilization. The technique recognizes that effective generation depends not only on isolated important tokens but also on maintaining the broader contextual architecture.

NACL [11] introduces a novel KV cache eviction framework that specifically addresses limitations in previous approaches by reconceptualizing when and how eviction occurs. Unlike traditional methods that perform eviction during the generation phase, NACL formulates eviction as a one-time operation during the encoding phase, progressively removing less important tokens layer by layer. This approach enables more efficient processing by leveraging global attention statistics rather than potentially biased local patterns. The core innovation of NACL lies in its Proxy-Tokens Eviction strategy, which identifies representative tokens from the query portion of inputs (typically found at the end of long documents) to guide the selection process. These proxy tokens naturally prioritize task-relevant information, resulting in more contextually appropriate retention decisions. By combining this proxy-based selection with progressive layer-wise eviction, NACL achieves superior memory efficiency while maintaining performance across diverse long-context tasks. This methodology demonstrates that effective cache management benefits from

considering both the global distribution of attention and the specific requirements of the current task.

Scissorhands [38] introduces an innovative approach to KV cache compression based on a fundamental insight about attention mechanisms in LLMs. The authors identify what they term the "Persistence of Importance Hypothesis," which observes that tokens demonstrating significance in previous computational steps tend to maintain their influence in subsequent operations. This temporal significance principle reveals that attention patterns across transformer layers exhibit high consistency—tokens that receive substantial attention early in processing continue to be pivotal throughout the network. By empirically demonstrating that this overlap exceeds 90% across most transformer layers, Scissorhands establishes a theoretical foundation for selective token retention. The method implements an efficient algorithm that dynamically compresses the KV cache while maintaining a predetermined memory budget, accompanied by theoretical guarantees that the compressed representation closely approximates full-attention outputs. This approach achieves 2-5× reduction in memory requirements without degrading model performance, enabling larger batch sizes during inference. Additionally, the authors demonstrate Scissorhands' compatibility with quantization techniques, highlighting its versatility as part of a comprehensive memory optimization strategy. This work represents a significant advancement in understanding how attention patterns can be leveraged to design more efficient inference systems for long-context processing.

Keyformer [1] presents an innovative solution to the context retention challenge faced by traditional attention mechanisms. While standard window attention methods suffer from semantic information loss by only retaining recent tokens, Keyformer implements a more nuanced selection strategy. Through empirical analysis, the authors discovered that a significant majority of attention distribution naturally concentrates on a minimal subset of tokens that carry disproportionate contextual value. Building on this insight, Keyformer employs a hybrid attention mechanism that strategically combines temporally proximate tokens with semantically significant ones identified throughout the sequence history. This approach preserves crucial contextual elements that would otherwise be discarded by fixed-window methods. Unlike previous techniques that rely solely on individual attention scores, Keyformer's token selection process incorporates relationship dynamics between tokens, enabling more informed preservation decisions. Experimental results demonstrate superior performance compared to contemporary approaches like H2O, particularly on tasks requiring long-range comprehension.

ActivationBeacon [62] introduces specialized beacon tokens that serve as information

carriers for long-context processing. Unlike traditional eviction strategies, this approach encodes contextual information into designated tokens' activations while discarding standard tokens after their information is distilled. The system employs a chunk-based workflow that processes input segments sequentially, preserving only beacon token activations between chunks. This enables processing of contexts significantly beyond the model's native window size (up to 128K tokens) while maintaining consistent compression ratios regardless of input length. Experimental results show 8× reduction in KV cache size and 2× acceleration in processing speed while preserving performance comparable to uncompressed baselines across various tasks.

ChunkKV [37] introduces a semantic-aware approach to KV cache compression that addresses a fundamental limitation in token-level selection methods. The authors identify that existing approaches evaluate token importance individually, neglecting the semantic interdependencies inherent in natural language. ChunkKV reformulates the compression challenge by treating semantically coherent token groups as fundamental compression units rather than isolated tokens. Through comprehensive analysis, the method identifies high similarity patterns in preserved chunk indices across transformer layers, enabling an efficient layer-wise index reuse mechanism that significantly reduces computational overhead. The framework implements a chunk-based observation window that aggregates attention scores within semantic units, preserving contextual relationships that would otherwise be fragmented through token-level evaluation. Experiments with instruction-tuned and multi-step reasoning models (O1 and R1) across diverse benchmarks—including LongBench, Needle-In-A-HayStack, GSM8K, and JailbreakV—demonstrate that ChunkKV achieves up to 10% performance improvement under aggressive compression scenarios compared to state-of-the-art methods. This semantic-aware approach proves particularly effective for complex reasoning tasks where preserving contextual coherence is essential for maintaining model capabilities.

SepLLM [9] leverages the observation that transformer models naturally prioritize attention to separator tokens (periods, commas, newlines) over semantic content tokens. This finding suggests these delimiters function as information compression points within attention mechanisms. SepLLM implements a data-dependent sparse attention framework that selectively preserves initial positions, neighboring tokens, and separators while pruning less important elements. The approach includes both training-free inference optimization and a training-integrated implementation with specialized hardware-efficient kernels. The latter reduces computational requirements by 28% and accelerates training by 26% while maintaining comparable performance to standard transformers.

4.1.3 Dynamic Selection without Permanent Eviction

Unlike permanent eviction methods, these techniques maintain all tokens in secondary storage while prioritizing a subset for active computation, balancing memory efficiency with information preservation through retrieval mechanisms.

InfLLM [57] is a training-free memory optimization framework designed for efficient processing of extremely long sequences. It employs a block-based KV cache mechanism that organizes token sequences into semantically coherent block units, retaining only tokens with high attention scores as representatives within each block. InfLLM implements a CPU-GPU hierarchical memory orchestration strategy, keeping frequently accessed blocks in GPU memory while offloading less frequently used blocks to CPU memory, thereby significantly reducing GPU memory consumption while maintaining processing performance. This approach not only enhances the effectiveness of relevant information retrieval but also improves computational efficiency by avoiding noisy context processing.

Quest [52] presents an innovative approach to KV cache optimization through query-aware token criticality estimation. Recognizing that token importance varies significantly depending on the query context, Quest dynamically identifies which portions of the KV cache are most relevant for each specific inference step. The framework implements a refined block representation technique based on minimal and maximal key values within KV cache blocks, enhancing retrieval precision while maintaining model accuracy. This selective attention mechanism allows for dramatic reductions in computational requirements, achieving up to 7.03× reduction in self-attention latency and 2.23× overall inference speedup compared to existing methods. Quest's effectiveness is demonstrated across various benchmarks including PG19, passkey retrieval tasks, and Long-Bench with contexts ranging from 256 to 32K tokens.

SqueezedAttention [24] introduces an efficient approach for accelerating LLM inference in applications with partially fixed input contexts. The method leverages offline K-means clustering to group keys based on semantic similarity, representing each cluster with a single centroid. During inference, SqueezedAttention performs query-centroid comparisons to selectively load only semantically relevant keys from the fixed context, substantially reducing computational and bandwidth requirements. Their approach extends to a hierarchical centroid lookup structure that reduces attention complexity from linear to logarithmic relative to context length. Evaluations across multiple long-context benchmarks demonstrate up to 4× speedup in both prefill and generation phases, with a 3× reduction in KV cache budget while maintaining accuracy (extending to 8× reduction with minimal accuracy impact). Their implementation includes optimized Triton kernels for

centroid comparison and sparse attention computation.

RetrievalAttention [33] addresses a significant challenge in utilizing Approximate Nearest Neighbor Search (ANNS) for attention computation: the out-of-distribution (OOD) problem between query and key vectors. The approach introduces a query-focused vector indexing strategy specifically designed for attention mechanisms, enabling efficient identification of critical tokens by traversing only 1-3% of key vectors while maintaining inference accuracy. RetrievalAttention implements a hybrid memory management system that keeps a minimal set of KV vectors in GPU memory based on static importance patterns while offloading the majority to CPU memory for index construction. During token generation, it efficiently retrieves critical tokens via ANNS indices on the CPU and merges partial attention results from both CPU and GPU computations. This dual-processing approach significantly reduces both inference latency and GPU memory requirements for long-context LLM applications.

EM-LLM [19] introduces a novel architecture that incorporates principles of event cognition and episodic memory into Transformer-based models. The approach dynamically segments input sequences into cohesive memory units based on the model's surprise signals during inference, followed by boundary refinement using graph-theoretic metrics derived from attention key similarities. This memory formation process requires minimal computational overhead (O(kn) where k is typically much smaller than the token count n). For retrieval, EM-LLM implements similarity-based mechanisms that preserve temporal contiguity and asymmetry effects, mimicking human-like patterns in sequential information retrieval. The system demonstrates superior performance over state-of-the-art retrieval models on long-context benchmarks and can successfully perform passkey retrieval across sequences of 10M tokens

ClusterKV [34] introduces a novel approach to KV cache compression that addresses the dynamic nature of token importance during the decoding process. Unlike traditional methods that permanently evict tokens based on fixed patterns or current attention weights, ClusterKV implements a semantic cluster-based recall mechanism. This approach recognizes that token importance changes throughout generation—tokens initially deemed unimportant may become critical in later decoding steps. By organizing tokens into semantic clusters rather than position-based pages, ClusterKV eliminates the internal fragmentation issues found in previous approaches like Quest, which recalled tokens at page granularity. The semantic clustering ensures that budget allocation is optimized for truly important tokens, improving both efficiency and model accuracy during long-context inference by maintaining a more semantically coherent compression strategy that better

aligns with the dynamic requirements of autoregressive generation.

4.2 KV Cache Budget Allocation

Budget allocation approaches distribute limited cache memory non-uniformly across model components, prioritizing resources based on empirical importance to output quality. These methods can target either attention heads or transformer layers.

4.2.1 Layer-wise Budget Allocation

These methods allocate different context lengths to various transformer layers, typically assigning longer contexts to early layers that capture local patterns and shorter contexts to later layers focused on higher-level abstractions.

PyramidInfer, PyramidKV [60, 7] both proposed novel approach to KV cache compression that challenges the conventional practice of using uniform cache sizes across all Transformer layers. Based on systematic analysis of attention distribution patterns, the authors identified a pyramid-like information flow where attention transitions from broad coverage of global contexts in lower layers to increasingly concentrated focus on key tokens in higher layers. Leveraging this insight, PyramidKV implements a graduated compression strategy that allocates larger KV cache budgets to lower layers where information is more dispersed, while progressively reducing cache sizes in higher layers where information becomes concentrated in fewer tokens. This layer-adaptive approach represents a significant departure from previous fixed-length methods such as StreamingLLM, SnapKV, and H2O, offering improved efficiency by aligning cache allocation with the actual informational requirements of each layer. This design enables PyramidInfer and PyramidKV to optimize memory usage while preserving model performance on long-context tasks.

DynamicKV [69] introduces a task-aware adaptive approach to KV cache compression that recognizes the varying memory requirements across different tasks and model layers. While previous methods like PyramidKV implement fixed compression patterns that decrease cache size with increasing layer depth, DynamicKV addresses the observation that different tasks exhibit distinct memory needs—summarization tasks require smaller KV caches in upper layers, whereas code completion demands larger upper-layer cache sizes. The method calculates attention scores between recent queries and context tokens, then dynamically determines optimal cache allocation through a correlation-based approach that gradually normalizes retention values. This adaptive strategy maintains temporary storage whose size is determined by task-specific requirements while minimizing memory overhead.

PrefixKV [54] introduces a novel KV cache compression approach based on the concept of retaining the most important "prefix" vectors from each layer's priority sequence. The method defines this prefix as the top-ranked KV vectors sorted by normalized importance scores, transforming the challenge of determining optimal layer-wise cache sizes into identifying the optimal global prefix configuration. PrefixKV leverages prefix cumulative priority as a measurement of preserved contextual information in each layer, then employs binary search to determine layer-specific retention ratios that collectively align with the overall compression budget while maximizing information preservation. This optimization strategy ensures that each transformer layer maintains maximal contextual information after compression, preserving generation quality even under significant memory constraints.

SimLayerKV [64] introduces a novel approach to KV cache reduction based on the observation that certain transformer layers exhibit "lazy" behavior—focusing primarily on semantically unimportant tokens such as initial tokens and recent ones during generation. Through extensive analysis, the authors demonstrate that these lazy layers contribute significantly less to long-context capabilities than non-lazy layers, making them ideal candidates for aggressive cache compression without substantial performance degradation. The method works by analyzing attention allocation patterns to identify lazy layers, then selectively trimming the KV cache only in those layers while preserving the full cache in non-lazy layers that perform critical information processing. This layer-adaptive approach avoids the pitfalls of uniform compression strategies that treat all layers equally. Notably, SimLayerKV requires minimal implementation effort (seven lines of code) while achieving 5× KV cache compression with only a 1.2% performance drop when combined with 4-bit quantization, maintaining robust performance even at extended context lengths up to 32K tokens.

4.2.2 Head-wise Budget Allocation

Head-wise allocation techniques recognize that not all attention heads contribute equally to model performance, allowing for selective pruning or compression of less critical heads to reduce memory requirements.

Ada-KV [16] addresses a critical limitation in existing KV cache eviction methods: their uniform budget allocation across attention heads fails to account for diverse concentration patterns, where some heads focus narrowly on small cache portions while others distribute attention broadly. This uniform approach creates inefficiencies—wasting cache budgets on heads with sparse concentration while incurring significant eviction losses in

heads with dispersed distribution. Ada-KV introduces the first adaptive budget allocation strategy that dynamically reallocates resources from heads with sparse concentration to those with more dispersed attention patterns. Guided by minimizing a theoretical upper bound of eviction loss between pre- and post-eviction attention outputs, the method offers a plug-and-play enhancement for existing Top-k eviction approaches. When integrated with state-of-the-art methods like SnapKV and PyramidKV, Ada-KV demonstrates significant performance improvements across both question-aware and question-agnostic compression scenarios on comprehensive benchmarks. The approach is particularly effective in challenging question-agnostic scenarios where compression is applied without leveraging question-specific information.

CriticalKV [17] advances KV cache optimization by expanding the notion of token importance beyond traditional attention weights to include value states and pretrained parameter matrices. This more comprehensive approach recognizes that the potential impact of tokens on model outputs depends on complex interactions across the entire network architecture. The framework introduces a principled selection algorithm that identifies and retains essential cache entries by minimizing the maximum potential output perturbation rather than relying solely on attention patterns. By accounting for these broader factors in importance assessment, CriticalKV achieves more robust performance preservation under aggressive cache reduction compared to methods that consider only attention-based importance signals.

LeanKV [65] presents a unified framework for KV cache optimization that integrates multiple compression strategies through three innovative principles. First, it introduces heterogeneous precision quantization (Hetero-KV), storing key vectors at higher precision than value vectors based on their differential impact on attention calculations. Second, it implements per-head dynamic sparsity that adapts memory allocation according to the varying importance distributions observed across both attention heads and requests. Third, it unifies these approaches into a comprehensive compression strategy where token importance serves as a unified metric for determining whether tokens should be stored at high precision, low precision, or pruned entirely. To overcome the implementation challenges of such fine-grained memory management, LeanKV introduces unified paging and on-GPU memory management that efficiently handles the variable precision requirements and token counts across different heads. This granular approach typically preserves 95% of the total attention mass while significantly reducing memory requirements, offering a more flexible and adaptive alternative to fixed-pattern compression methods that don't account for the distinctive characteristics of individual attention mechanisms.

RazorAttention [51] introduces a novel perspective on KV cache compression by identifying a fundamental "retrieve and process" mechanism in LLMs. Unlike previous approaches that indiscriminately drop tokens based on importance scores, RazorAttention recognizes that certain attention heads—termed "retrieval heads"—play a crucial role in recalling relevant information from the input context, while "non-retrieval heads" primarily focus on local context processing. This insight enables a head-specific compression strategy: maintaining complete KV cache for retrieval heads (including both echo heads that focus on identical tokens and induction heads that attend to antecedent tokens) while significantly compressing non-retrieval heads. For these compressed heads, RazorAttention introduces an innovative "compensation token" that encapsulates information from dropped tokens, preserving semantic content while reducing memory requirements. This approach successfully compresses up to 70% of the KV cache without notable performance degradation. Importantly, unlike previous importance-based methods, RazorAttention maintains full compatibility with FlashAttention since it doesn't rely on attention maps for token selection, enabling substantial inference acceleration in practical applications.

HeadKV [20] advances beyond previous head-categorization approaches like RazorAttention by recognizing that effective long-context processing requires both information retrieval and complex reasoning capabilities. Rather than simply identifying and prioritizing retrieval heads, HeadKV introduces a sophisticated assessment framework that jointly evaluates each attention head's contributions to both retrieval and reasoning tasks. This dual-capability evaluation uses specially constructed prompts that require the model to first extract factual information from the context and then perform logical operations on that information. The resulting importance scores guide fine-grained budget allocation at the individual head level, with higher-scoring heads receiving larger KV cache allocations. Within each head, further optimization occurs by retaining only the most relevant KV cache entries. This nuanced approach demonstrates superior performance on challenging benchmarks like needle-in-a-Haystack and reasoning-in-a-Haystack tests, effectively preserving model capabilities on tasks that demand both precise information extraction and sophisticated reasoning while significantly reducing memory requirements and decoding latency.

DuoAttention [58] introduces an efficient approach to long-context processing based on the key observation that attention heads in LLMs naturally separate into two distinct functional categories. Unlike previous methods that treat all heads uniformly, DuoAttention employs a parameterized, optimization-based procedure that distinguishes between retrieval heads (requiring full context attention) and streaming heads (which func-

tion effectively with limited context). Rather than analyzing attention patterns, DuoAttention directly measures output deviation resulting from token dropping to identify non-compressible heads, leading to more accurate categorization. The framework implements a dual-cache architecture where each transformer layer maintains separate memory structures: a complete KV cache for the identified retrieval heads and a compressed constant-size cache (containing only attention sinks and recent tokens) for streaming heads. This targeted approach substantially reduces memory requirements while maintaining model accuracy, achieving speedups of up to 2.55× for MHA models and 1.67× for GQA models. Importantly, DuoAttention maintains full compatibility with other optimization techniques like grouped-query attention and quantization, making it a versatile solution for resource-constrained deployment scenarios.

CHAPTER 5

CONTRIBUTIONS TO EFFICIENT LONG-CONTEXT PROCESSING

This chapter presents my contributions to the field of efficient long-context processing in Large Language Models (LLMs), with particular emphasis on KV Cache optimization techniques. The research addresses three key challenges: (1) developing more effective evaluation frameworks for assessing long-context generation capabilities, (2) introducing novel approaches to KV Cache compression that improve upon existing methods, and (3) evaluating the influence of KV Cache compression methods on LLMs' fundamental abilities.

5.1 LongGenBench: A Benchmark for Long-Context Generation

Abstract: Long-context capabilities have become increasingly important in advancing Large Language Models (LLMs). However, most existing benchmarks such as needle-in-a-haystack (NIAH) primarily focus on retrieval-based tasks, which only require models to locate and extract specific information from extensive contexts, like Figure 5.1 (a) and (b). These benchmarks fail to evaluate a crucial aspect of long-context processing: the ability to generate coherent and contextually accurate text that spans across lengthy passages or documents.

Algorithm 1 Pipeline of **LongGenBench**

Input: System Prompt S, Questions Q, Number of Questions K, Number of Iterations T, Language Model LLM
Output: Long-Context Responses R
$R \leftarrow \emptyset$
for $t \leftarrow 0$ to $T-1$ **do**
    $Q_t \leftarrow Q[t \times K : (t+1) \times K]$
    $\mathtt{InputPrompt} \leftarrow S + \text{concatenate}(Q_t)$
    $\mathtt{Response} \leftarrow \text{LM.gen}(\mathtt{InputPrompt})$
    $\mathtt{ParsedResponse} \leftarrow \text{parse}(\mathtt{Response})$
    $R \leftarrow R \cup \{\mathtt{ParsedResponse}\}$
    $\text{verify}(\mathtt{ParsedResponse}, Q_t)$
Output: R

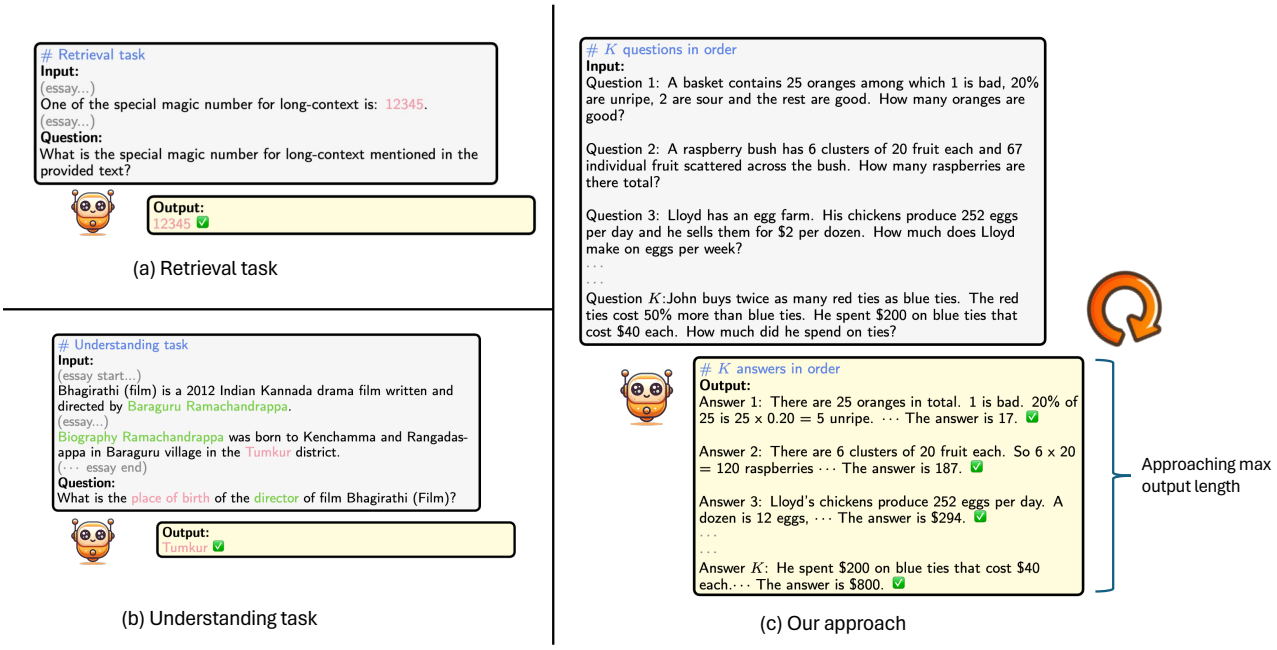


Figure 5.1. Illustrations of previous long-context benchmarks and our proposed approach. **(a) Retrieval task**: requires LLMs to retrieve the magic information hidden within an unrelated long context. **(b) Understanding task**: requires LLMs to comprehensively understand a long essay and answer the specific question. **(c) Our approach**: reconstructs the format of the dataset, requiring LLMs to sequentially understand and respond to each question in a single response. We run multiple iterations with different questions to evaluate the robustness of long-context generation capabilities. The length of the generated responses aims to approach the token limit.

Method: To address this significant gap, I introduce LongGenBench [35], a novel synthetic benchmark specifically designed to evaluate the long-context generation capabilities of LLMs. Unlike traditional retrieval-focused benchmarks, LongGenBench fundamentally redesigns the evaluation paradigm by:

- Focusing on consistency in logical flow across long-form generations
- Restructuring question formats to necessitate single, cohesive long-context answers
- Providing a comprehensive assessment framework that targets generation rather than mere information retrieval

The Algorithm 1 gives a pseudocode outline for the LongGenBench. The system prompt S contains instructional information, while Q is a list of questions from the original dataset. For each iteration t, a batch of K questions, Q_t, is selected from Q within the range $[t \times K : (t+1) \times K]$. The selected questions are concatenated to the system S to form the $\mathtt{InputPrompt}$. The language model LLM generates a long-context response for the given $\mathtt{InputPrompt}$. The response is added to the response set R, then parsed and verified for correctness and sequence. This process is repeated for T iterations, with each iteration featuring a unique

set of questions. The final output is the set of long-context responses R.

Experiment Results: Through extensive evaluation using LongGenBench, several key insights emerged:

1. Both API-accessed and open-source models exhibit significant performance degradation in long-context generation scenarios, with degradation rates ranging from 1.2% to 47.1%
2. Different model families demonstrate varying trends in performance degradation, suggesting architectural differences affect long-context generation abilities
3. Among API-accessed models, Gemini-1.5-Flash demonstrated the least degradation, while the QWEN2 series exhibited the strongest performance among open-source models

Model	GSM8K (%)		
	Baseline ↑	LongGenBench ↑	Delta Δ
GPT-3.5-Turbo	75.1	55.3	-19.8∇
GPT-4o	91.1	75.6	-15.5∇
Gemini-1.5-Flash	86.2	85.0	-1.2∇
Claude-3-Haiku	76.6	55.3	-21.3∇

Model	MMLU (%)		
	Baseline ↑	LongGenBench ↑	Delta Δ
GPT-3.5-Turbo	70.0	56.3	-13.7∇
GPT-4o	88.7	79.7	-9.0∇
Gemini-1.5-Flash	79.0	74.7	-4.3∇
Claude-3-Haiku	75.0	48.6	-26.4∇

Table 5.1. Comparison of baseline and LongGenBench performance on GSM8K (top) and MMLU (bottom) datasets with API-accessed models.

The data reveals nuanced relationships between baseline performance and long-context generation capabilities. Among API-accessed models (Table 5.1), Gemini-1.5-Flash demonstrates exceptional resilience with minimal degradation on both GSM8K (-1.2%) and MMLU (-4.3%), despite not having the highest baseline scores. In contrast, Claude-3-Haiku shows the most severe performance drops on both benchmarks (-21.3% and -26.4% respectively), suggesting architectural limitations in long-form generation. For open-source models (Table 5.2), Qwen2-72B-Instruct and DeepSeek-v2-Chat maintain consistent performance with minimal degradation across both datasets (approximately -5.5% on GSM8K and -6% on MMLU). Most notably, models with identical baseline scores exhibit dramatically different degradation patterns—LLaMA-3-8B-Instruct, Qwen2-57B-A14B-Instruct, and ChatGLM4-9B-Chat all score 79.6% on the GSM8K baseline (Table 5.2), yet their performance drops

Model	GSM8K (%)		
	Baseline↑	LongGenBench↑	Delta∆
LLaMA-3-8B-Instruct	79.6	32.5	-47.1∇
LLaMA-3-70B-Instruct	93.0	83.2	-9.8∇
Qwen2-7B-Instruct	82.3	63.9	-18.4∇
Qwen2-57B-A14B-Instruct	79.6	71.2	-8.4∇
Qwen2-72B-Instruct	91.1	85.7	-5.4∇
ChatGLM4-9B-Chat	79.6	68.8	-10.8∇
DeepSeek-v2-Chat	92.2	86.5	-5.7∇

Model	MMLU (%)		
	Baseline↑	LongGenBench↑	Delta∆
LLaMA-3-8B-Instruct	68.4	50.4	-18.0∇
LLaMA-3-70B-Instruct	82.0	71.2	-10.8∇
Qwen2-7B-Instruct	70.5	59.4	-11.1∇
Qwen2-57B-A14B-Instruct	75.4	66.7	-8.7∇
Qwen2-72B-Instruct	82.3	75.8	-6.5∇
ChatGLM4-9B-Chat	72.4	63.0	-9.4∇
DeepSeek-v2-Chat	77.8	72.0	-5.8∇

Table 5.2. Comparison of baseline and LongGenBench performance on GSM8K and MMLU datasets with open source models.

differ by factors of 4-5× (-47.1%, -8.4%, and -10.8% respectively). Similarly, LLaMA-3-70B-Instruct achieves the highest GSM8K baseline (93.0%) but experiences a substantial -9.8% drop, while GPT-4o (Table 5.1) shows a larger -15.5% decline from its 91.1% baseline. These variations highlight that raw performance on standard benchmarks does not necessarily predict a model's capacity for sustained reasoning in long-context generation scenarios.

These findings highlight that even models performing well on traditional retrieval-based long-context benchmarks may struggle significantly with long-form generation tasks. This discrepancy underscores the importance of comprehensive evaluation frameworks that capture the multifaceted nature of long-context processing. Additionally, currently popular thinking-oriented LLMs (such as O1/R1) also exhibit similar performance degradation in long-text generation tasks.

The LongGenBench framework and associated resources are publicly available at `https://github.com/Dominic789654/LongGenBench`, enabling researchers and practitioners to conduct standardized evaluations of long-context generation capabilities.

Conclusion: In this study, we introduced LongGenBench, an effective framework designed to evaluate the long-context generation capabilities of language models (LLMs) across multiple datasets. Our experiments included both API accessed and open source

models, offering a comprehensive comparison of their performance in long-context generation tasks. The results indicate a correlation between baseline performance and LongGenBench performance, with higher baseline models generally exhibiting smaller declines. Additionally, model size and architecture significantly influence resilience, with larger models and specific architectures demonstrating greater robustness and consistent trends across different LongGenBench tasks. These findings highlight the importance of considering both model architecture and size when evaluating LLMs in long-context generation tasks. The LongGenBench framework effectively showcases the varying capabilities of different models, providing valuable insights for further model development and optimization.

5.2 ChunkKV: Semantic-Aware Chunk-wise KV Cache Compression

Abstract: Large Language Models (LLMs) encounter significant memory constraints when processing extended contexts, with the Key-Value (KV) cache being a primary resource bottleneck. While numerous compression techniques have emerged to address this challenge, we identify a critical limitation in existing approaches: they predominantly evaluate token importance in isolation, failing to account for the semantic interdependencies that characterize natural language structure.

In response to this limitation, we introduce ChunkKV, a novel compression framework that treats semantically coherent token groups as fundamental compression units rather than individual tokens. By preserving or discarding these semantic chunks holistically, ChunkKV maintains crucial contextual relationships that would otherwise be fragmented through token-level evaluation. Furthermore, our analysis reveals substantial similarity patterns in preserved chunk indices across transformer layers, enabling an efficient layer-wise index reuse mechanism that significantly reduces computational overhead.

We conducted comprehensive evaluations across diverse benchmarks, including established long-context assessments (LongBench [4], Needle-In-A-HayStack [29]) and challenging reasoning tasks (GSM8K [13], JailbreakV [40]). Experiments with advanced instruction-tuned and multi-step reasoning models (O1 [26] and R1 [22]) demonstrate that ChunkKV achieves up to 10% performance improvement under aggressive compression scenarios compared to state-of-the-art methods. These results validate that semantic-aware chunking strategies more effectively preserve model capabilities while substantially reducing memory requirements, advancing the practical deployment of LLMs in resource-constrained environments.

Algorithm 2 ChunkKV

Input: $\mathbf{Q} \in \mathbb{R}^{T_q \times d}$, $\mathbf{K} \in \mathbb{R}^{T_k \times d}$, $\mathbf{V} \in \mathbb{R}^{T_v \times d}$, observe window size w, chunk size c, compressed KV cache max length L_{max}
Output: Compressed KV cache $\mathbf{K}'$, $\mathbf{V}'$
Observe Window Calculation:
$\mathbf{A} \leftarrow \mathbf{Q}_{T_q-w:T_q}\mathbf{K}^T$ {Attention scores for the observe window}
$C \leftarrow \lceil \frac{T_k}{c} \rceil$ {Calculate the number of chunks}
Chunk Attention Score Calculation:
for $i = 1$ to C **do**
    $\mathbf{A}_i \leftarrow \sum_{j=(i-1)c+1}^{ic} \mathbf{A}_{:,j}$ {Sum of observation scores for each chunk}
Top-K Chunk Selection:
$k \leftarrow \lfloor \frac{L_{max}}{c} \rfloor$
Top_K_Indices $\leftarrow$ indices of Top-k chunks based on $\mathbf{A}_i$
Compression:
$\mathbf{K}', \mathbf{V}' \leftarrow$ index_select($\mathbf{K}, \mathbf{V}$, *Top_K_Indices*)
Concatenation:
$\mathbf{K}' \leftarrow \text{concat}(\mathbf{K}'_{0:L_{max}-w}, \mathbf{K}_{T_k-w:T_k})$
$\mathbf{V}' \leftarrow \text{concat}(\mathbf{V}'_{0:L_{max}-w}, \mathbf{V}_{T_v-w:T_v})$
$\mathbf{K}'$, $\mathbf{V}'$

Methods: The Algorithm 2 shows the pseudocode outline of ChunkKV. First, following H2O [66] and SnapKV [32], we set the observe window by computing the observation scores $\mathbf{A} \leftarrow \mathbf{Q}_{T_q-w:T_q}\mathbf{K}^T$, where $\mathbf{Q}_{T_q-w:T_q}$ is the observe window, $\mathbf{K}$ is the Key matrix and the window size w is usually set to $\{4, 8, 16, 32\}$. Next, the number of chunks C is calculated as $C = \lceil \frac{T_k}{c} \rceil$, where T_k is the length of the Key matrix and c is the chunk size. The observation scores for each chunk are then computed as $\mathbf{A}_i = \sum_{j=(i-1)c+1}^{ic} \mathbf{A}_{:,j}$ for $i = 1, 2, \ldots, C$. Referring to previous works [66, 32, 60, 7], we still use the top-k algorithm as ChunkKV's sampling policy. For the top-k chunk selection, the top-k chunks are selected based on their observation scores, where $k = \lfloor \frac{L_{max}}{c} \rfloor$, and L_{max} is the maximum length of the compressed KV cache. The size of the last chunk will equal $\min(c, L_{max} - (k-1) \times c)$. The indices of the top-k chunks will keep the original sequence order. In the compression step, the key and value matrices are only retained based on the selected indices, resulting in the compressed KV cache. Finally, the observe window of the original KV cache will be concatenated to the compressed KV cache by replacing the last w tokens to keep important information. The compressed KV cache is then used for subsequent attention computations.

Experiment Results: In this section, we will evaluate the performance of ChunkKV on the GSM8K, LongBench and Needle-In-A-HayStack dataset.

GSM8K Results. Table 5.3 presents the performance comparison. The results demonstrate that ChunkKV outperforms other KV cache compression methods on different models and compression ratios. Table 5.4 presents the performance comparison of many-shot

Table 5.3. GSM8K Performance Comparison.

Ratio	StreamingLLM	H2O	SnapKV	PyramidKV	**ChunkKV (Ours)**
DeepSeek-R1-Distill-Llama-8B FullKV: 69.4% ↑					
10%	51.6%	55.6%	57.6%	62.6%	**65.7%**
LlaMa-3.1-8B-Instruct FullKV: 79.5% ↑					
30%	70.5%	72.2%	76.1%	77.1%	**77.3%**
20%	63.8%	64.0%	68.8%	71.4%	**77.6%**
10%	47.8%	45.0%	50.3%	48.2%	**65.7%**
LlaMa-3-8B-Instruct FullKV: 76.8% ↑					
30%	70.6%	73.6%	70.2%	68.2%	**74.6%**
Qwen2-7B-Instruct FullKV: 71.1% ↑					
30%	70.8%	61.2%	70.8%	64.7%	**73.5%**

Table 5.4. Many-Shot GSM8K Performance Comparison.

Ratio	StreamingLLM	H2O	SnapKV	PyramidKV	**ChunkKV (Ours)**
DeepSeek-R1-Distill-Llama-8B FullKV: 71.2% ↑					
10%	63.2%	54.2%	54.1%	59.2%	**68.2%**
LlaMa-3.1-8B-Instruct FullKV: 82.4% ↑					
10%	74.3%	51.2%	68.2%	70.3%	**79.3%**

GSM8K, also ChunkKV outperforms other KV cache compression methods. The consistent superior performance of ChunkKV in both models underscores its effectiveness in maintaining crucial contextual information for complex arithmetic reasoning tasks.

Table 5.5. KV cache compression methods on the LongBench benchmark. Results show performance gap compared to FullKV baseline (negative values indicate worse performance).

Ratio	StreamingLLM	H2O	SnapKV	PyramidKV	**ChunkKV (Ours)**
LlaMa-3-8B-Instruct FullKV: 41.46 ↑					
10%	-13.80%	-10.61%	-3.16%	-3.33%	**-2.29%**
20%	-6.42%	-8.85%	-2.24%	-2.00%	**-1.74%**
30%	-2.36%	-5.38%	-0.07%	-0.22%	**+0.31%**
Mistral-7B-Instruct-v0.3 FullKV: 48.08 ↑					
10%	-16.58%	-9.30%	-3.54%	-3.52%	**-2.85%**
Qwen2-7B-Instruct FullKV: 40.71 ↑					
10%	-5.28%	-0.64%	-0.39%	-0.98%	**+0.42%**

LongBench Results. Tables 5.5 show that ChunkKV is capable of achieving on-par performance or even better than the full KV cache with less GPU memory consumption. This table presents the performance gap (in percentage) between each method and the FullKV baseline, where negative values indicate performance degradation compared to FullKV. The table is evaluated in the LongBench English subtask, where ChunkKV outperforms other compression methods overall. This suggests that ChunkKV's approach of retaining semantic chunks is more effective in preserving important information compared to other discrete token-based compression methods.

Table 5.6. NIAH Performance Comparison.

KV cache Size	StreamingLLM	H2O	SnapKV	PyramidKV	**ChunkKV (Ours)**
LlaMa-3.1-8B-Instruct FullKV: 74.6% ↑					
512	32.0%	68.6%	71.2 %	72.6%	**74.5%**
256	28.0%	61.7%	68.8%	69.5%	**74.1%**
128	23.7%	47.9%	58.9%	65.1%	**73.8%**
96	21.5%	41.0%	56.2%	63.2%	**70.3%**
Mistral-7B-Instruct FullKV: 99.8% ↑					
128	44.3%	88.2%	91.6%	99.3%	**99.8%**

Needle-In-A-HayStack Results. Table 5.6 provides statistical results for different compression methods. These findings clearly indicate the effectiveness of ChunkKV in managing varying token lengths and depth percentages, making it a robust choice for KV cache management in LLMs. Figure 5.6 presents the NIAH benchmark results for LLaMA-3-8B-Instruct. The vertical axis represents the depth percentage, while the horizontal axis represents the token length, with shorter lengths on the left and longer lengths on the right. A cell highlighted in green indicates that the method can retrieve the needle at that length and depth percentage.

Conclusion We introduced ChunkKV, a novel KV cache compression method that preserves semantic information by retaining more informative chunks. Through extensive experiments across multiple state-of-the-art LLMs (including DeepSeek-R1, LLaMA-3, Qwen2, and Mistral) and diverse benchmarks (GSM8K, LongBench, NIAH, and JailbreakV), we demonstrate that ChunkKV consistently outperforms existing methods while using only a fraction of the memory. Our comprehensive analysis shows that ChunkKV's chunk-based approach maintains crucial contextual information, leading to superior performance in complex reasoning tasks, long-context understanding, and safety evaluations. The method's

effectiveness is particularly evident in challenging scenarios like many-shot GSM8K and multi-document QA tasks, where semantic coherence is crucial. Furthermore, our proposed layer-wise index reuse technique provides significant computational efficiency gains with minimal performance impact, achieving up to 20.7% latency reduction and 26.5% throughput improvement. These findings, supported by detailed quantitative analysis and ablation studies, establish ChunkKV as a significant advancement in KV cache compression technology, offering an effective solution for deploying LLMs in resource-constrained environments while maintaining high-quality outputs.

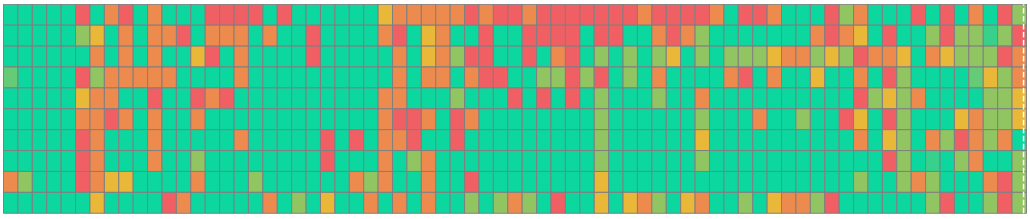


Figure 5.2. ChunkKV, accuracy 73.8%

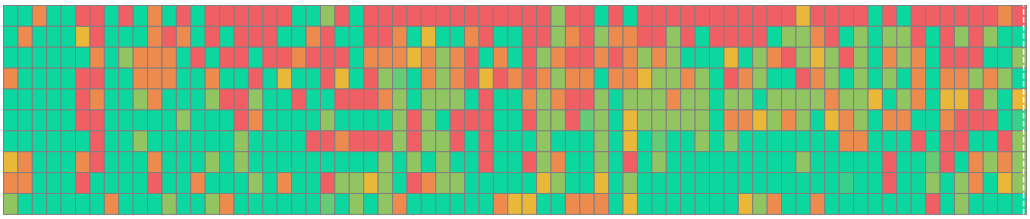


Figure 5.3. PyramidKV, accuracy 65.1%

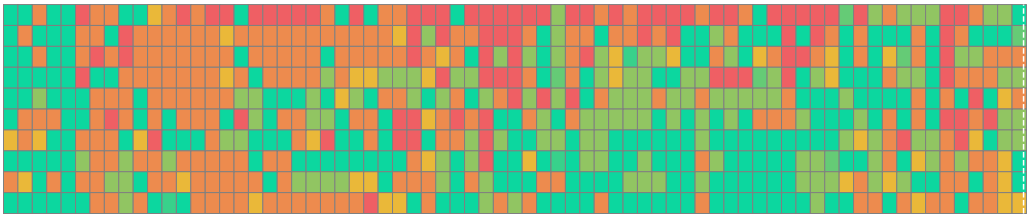


Figure 5.4. SnapKV, accuracy 58.9%

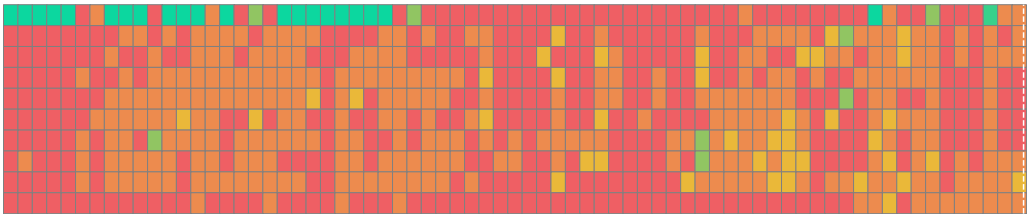


Figure 5.5. StreamingLLM, accuracy 23.7%

Figure 5.6. NIAH benchmark for LLaMA3-8B-Instruct with KV cache size=128 under 8k context length.

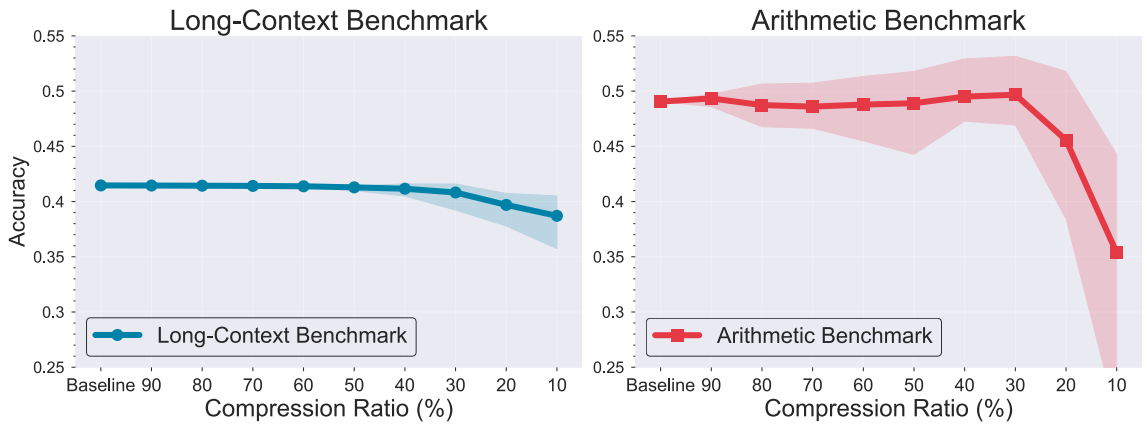
5.3 Can LLMs Maintain Fundamental Abilities under KV Cache Compression?

Abstract: While KV cache compression techniques have demonstrated effectiveness for long-context processing, their impact on LLMs' fundamental capabilities remains underexplored. This section presents a comprehensive empirical study examining how various compression methods affect core model abilities across diverse task categories.

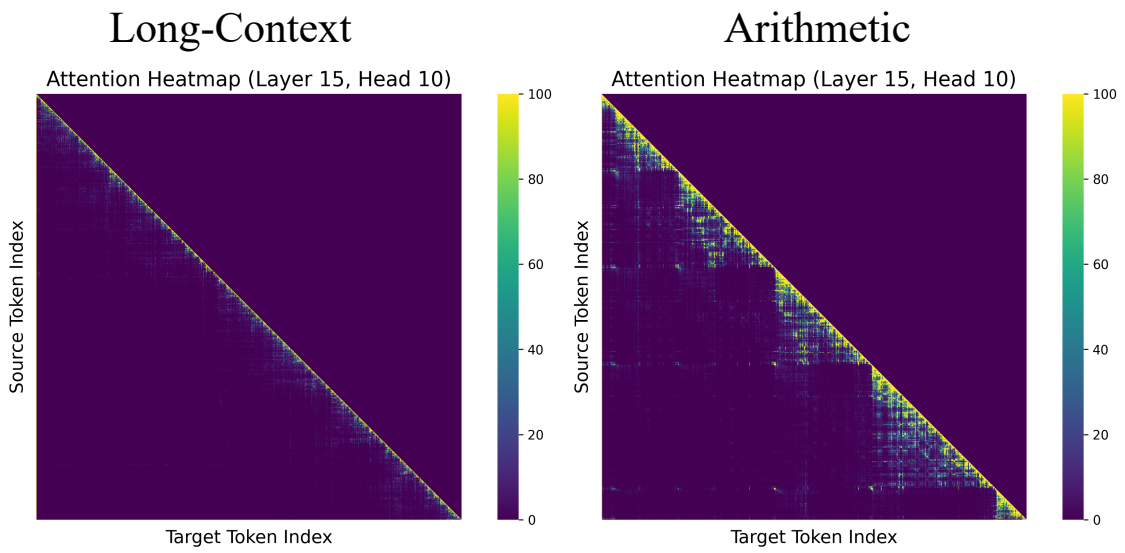
Our preliminary analysis, as shown in Figure 5.7, reveals two critical findings: (a) arithmetic reasoning tasks suffer significantly higher performance degradation under compression compared to long-context tasks, and (b) attention patterns in long-context scenarios exhibit notably higher sparsity than arithmetic tasks. These observations suggest that current evaluation frameworks, which focus predominantly on long-context performance, may inadequately capture the full spectrum of impacts on model capabilities. Further analysis of attention patterns, visualized in Figure 5.8, reveals distinct task-specific behaviors: **while world knowledge and commonsense reasoning tasks exhibit universal attention distributions, arithmetic reasoning and safety tasks demonstrate more specialized patterns**. Specifically, arithmetic reasoning tasks display increased attention sparsity, indicating focused attention on individual prompt examples, while safety tasks show concentrated attention on the system prompt. In contrast, world knowledge and commonsense reasoning tasks demonstrate more uniform attention distribution across the entire prompt. These varying attention patterns—visualized through colored squares representing system prompts, shot examples, and questions—provide insights into task-specific context utilization and motivate our investigation into how compression affects various factors including model size, prompt length, and task type.

Methodology We evaluate prominent KV cache compression techniques (StreamingLLM [59], H2O [66], SnapKV [32], PyramidKV [7], ChunkKV [37]) on instruction-tuned models (LLaMA-3.1-8B-Instruct, LLaMA-3-8B [15]) and reasoning-focused models (DeepSeek-R1-Distill [22]). Our evaluation spans six capability domains:

- **World Knowledge:** Using factual retrieval benchmarks to assess preservation of encyclopedic knowledge, MMLU [23]
- **Commonsense Reasoning:** Evaluating logical inference capabilities on everyday scenarios, CommonSenseQA [50]
- **Arithmetic Reasoning:** Testing mathematical problem-solving using GSM8K benchmark [13]
- **Code Generation:** Assessing ability to produce syntactically correct and functional



a) KV Cache compression methods on long-context and arithmetic benchmarks



b) Attention heatmap on long-context and arthmetic benchmarks

Figure 5.7. KV cache compression methods on long-context and arithmetic benchmarks. (a) Arithmetic benchmark shows more performance degradation than long-context benchmark. (b) Long-Context benchmark shows more sparsity in attention heatmap.

code, HumanEval [10]

- **Safety:** Measuring alignment with safety guidelines using JailbreakV benchmark [40]
- **Long-Context Generation:** Evaluating long-context generation using LongGenBench [35]

For each capability domain, we measure performance under varying compression ratios (10%, 20%, 30%, 40%, 50%) compared to full KV cache baselines.

Results This section presents the results of our comprehensive evaluation of KV cache compression methods across various tasks and models.

Observation 1. **KV cache compression methods show task-dependent performance degradation, with varying sensitivity thresholds across different benchmark categories.** As demonstrated in Figure 5.9, all tasks maintain stable performance at compression ratios above 40%, but exhibit distinct degradation patterns below this threshold. GSM8K, Hu-

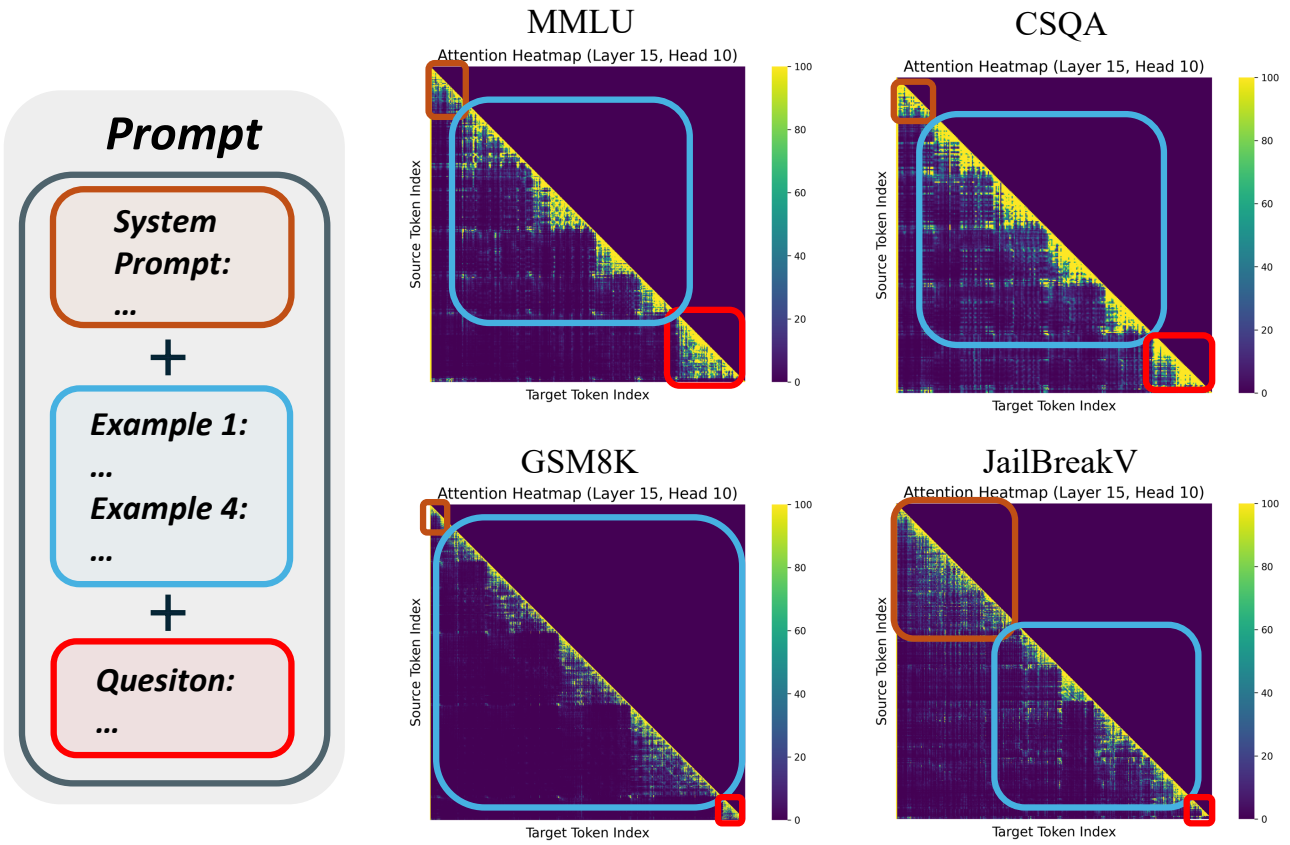
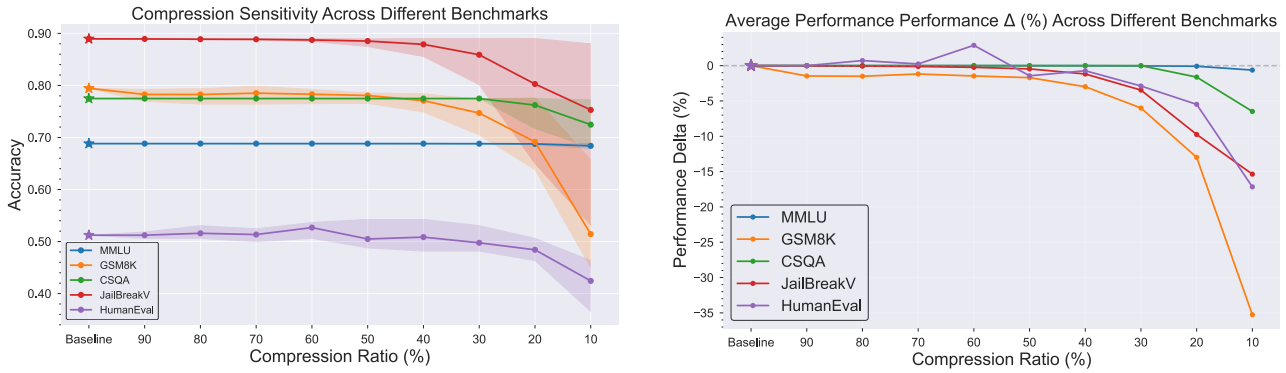


Figure 5.8. Attention heatmap on different tasks. The heatmap is generated by the attention scores of the 15-th layer of the LLaMA-3.1-8B-Instruct attention head 10.

manEval, and JailBreakV tasks demonstrate the highest compression sensitivity, characterized by precipitous performance declines. Figure 5.10 illustrates the detailed performance impact of various KV cache compression methods across different tasks. This degradation is most pronounced in GSM8K (d), where performance deteriorates significantly below 20% compression ratio, with accuracy dropping from approximately 0.75 to below 0.5. Among the evaluated methods, ChunkKV [37] and PyramidKV [7] consistently demonstrate superior stability across most tasks, while StreamingLLM [59] exhibits heightened sensitivity to aggressive compression. Additionally, R1-GSM8K (e) indicates that R1 LLMs demonstrate enhanced robustness to KV cache compression.

Observation 2. **Multi-step reasoning LLMs are more robust to KV cache compression.** Figure 5.11 presents a comparative analysis of LLaMA-3.1-8B across its base (w/o instruct tuned), instruct-tuned, and DeepSeek-R1 distilled variants, illustrating their averaged performance across five compression methods with confidence intervals. Although all three variants exhibit performance degradation at low compression ratios, their degradation trajectories differ significantly. The R1 distilled model demonstrates superior stability, maintaining performance around 0.60 even at a 10% compression ratio. While the instruct-tuned model achieves a higher initial accuracy (0.8), it exhibits heightened compression



a) Sensitivity Analysis of Different Benchmark Categories to KV Cache Compression

b) Performance Delta Lines with Baseline

Figure 5.9. Sensitivity Analysis of Different Benchmark Categories to KV Cache Compression.

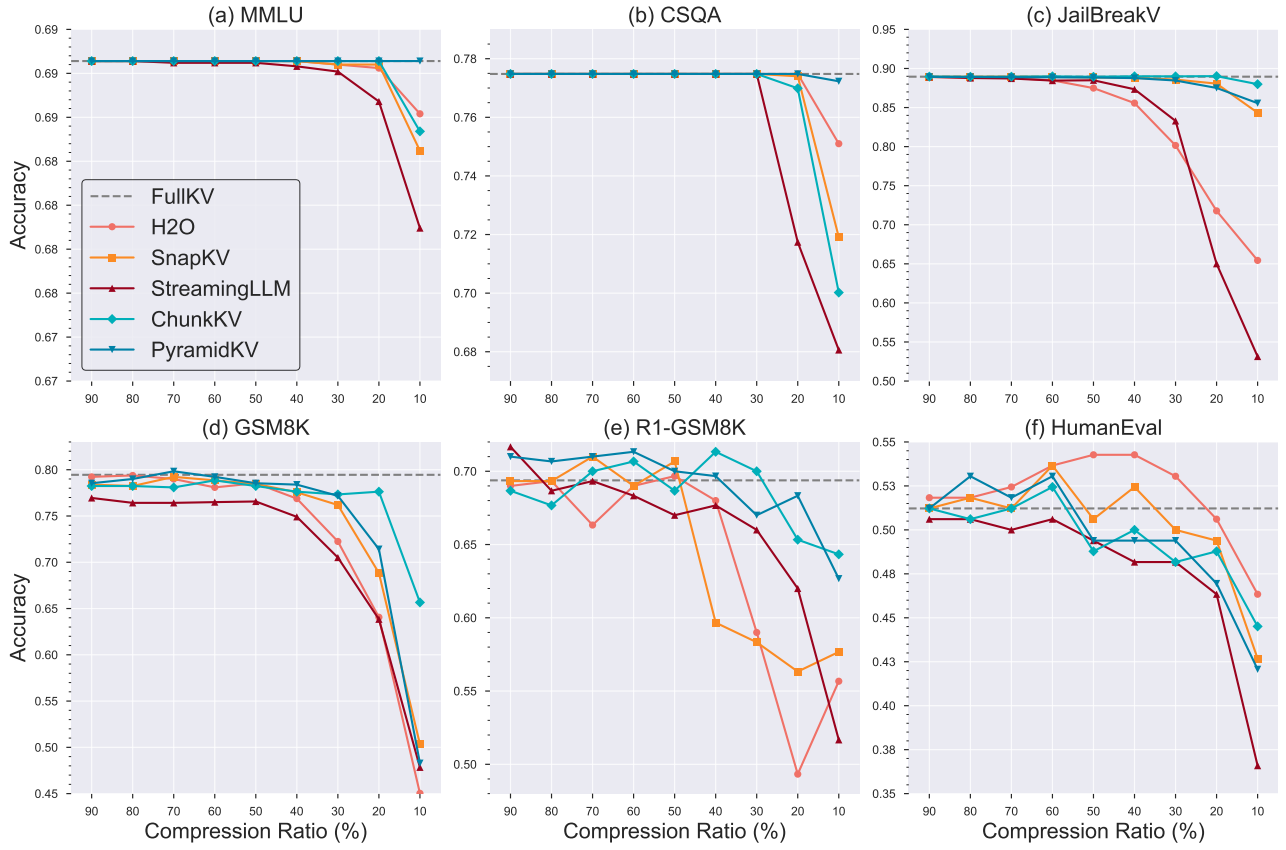


Figure 5.10. Performance Comparison of KV Cache Compression Methods Across Tasks. Note: The y-axis scales vary across different tasks. Results for R1-GSM8K (e) were obtained using the DeepSeek-R1-Distill-Llama-8B model.

sensitivity, with performance deterioration initiating at 30% compression ratio and declining sharply to approximately 0.5 at 10% ratio. These findings suggest that while multi-step reasoning LLMs demonstrate enhanced robustness to KV cache compression, and instruct-tuning improves overall model performance, the latter may inadvertently increase model vulnerability to aggressive compression, particularly at compression ratios below

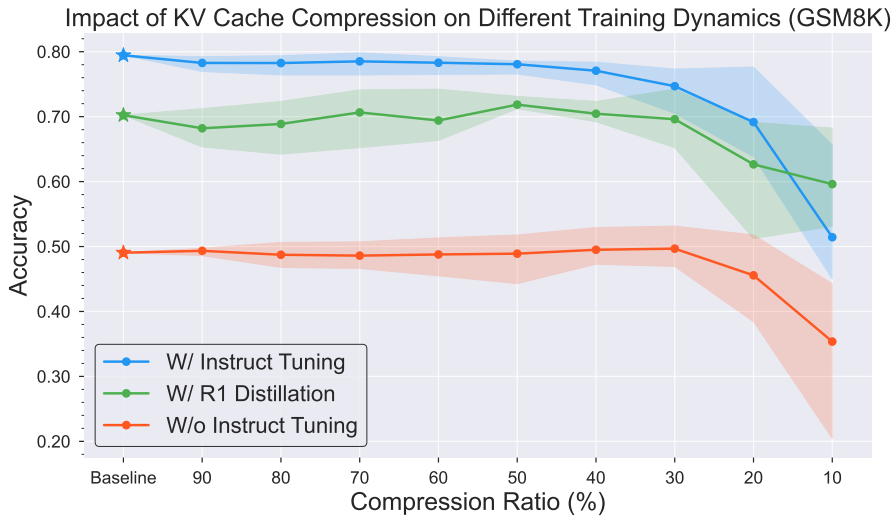
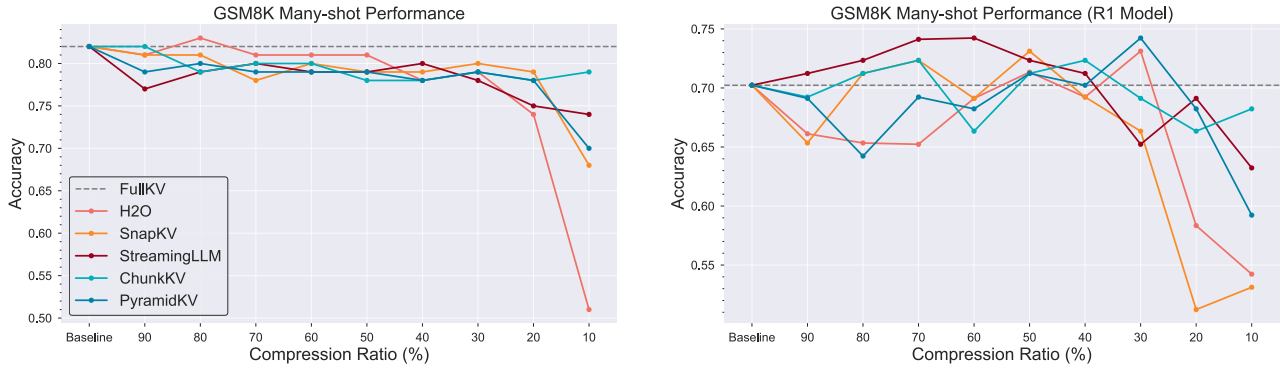


Figure 5.11. Performance Comparison of KV Cache Compression Methods on different training dynamics on GSM8K

30%.

Observation 3. **Chunk-level compression is more effective for long-context arithmetic reasoning tasks.** Inspired by [2], we consider many-shot in-context learning as a long-context reasoning task, which is more complex than existing long-context benchmarks, such as LongBench and NIAH. Figure 5.12 shows the performance of KV cache compression methods on a 50-shot GSM8K task, where the prompt length exceeds 4K tokens. From the figure, we observe that ChunkKV [37] demonstrates the most stability when the compression ratio is below 10% on both LLaMA-3.1-8B-Instruct and DeepSeek-R1-Distill-Llama-8B, indicating that in more complex long-context arithmetic reasoning tasks, chunk-level retention is more effective at preserving semantic information.



a) Many-shot GSM8K on LLaMA3.1-8B-Instruct

b) Many-shot GSM8K on DeepSeek-R1-Distill-Llama-8B

Figure 5.12. Many-shot scenario on KV cache compression

ShotKV Based on our comprehensive analysis, we find that current unified KV cache compression methods exhibit significant performance degradation on Long-Context Generation tasks. In this section, we introduce ShotKV, a decoding-time compression method

designed to mitigate this degradation. Our approach is motivated by two key insights: (1) Figure 5.8 demonstrates that n-shot example prompts receive substantial attention in reasoning benchmarks, and (2) Observation 4 reveals that chunk-level compression is particularly effective for long-context arithmetic reasoning tasks. Based on these findings, we hypothesize that each shot example represents a coherent chunk of information, leading us to design ShotKV to preserve shot examples intact during decoding time.

Implementation The **ShotKV** (Prefill-Decoding Separated **Shot**-aware **KV** Cache Compression), which separates the compression strategy for prefill and decoding phases. The key insight is that the prefill phase KV cache, which contains crucial prompt information, should be compressed once and remain fixed, while the decoding phase KV cache can be dynamically compressed with different strategies.

Experiments From the Table 5.7, we can see that ShotKV achieves the best performance on LongGenBench-GSM8K, maintaining high performance at low compression ratios. Specifically, at a compression ratio of 40%, ShotKV achieves 47.33% accuracy, surpassing the full kv cache baseline (46.00%) and showing substantial improvements over other methods (32.66%-39.50%). This superior performance can be attributed to two key design choices: (1) the preservation of complete shot examples during compression maintains the semantic coherence necessary for mathematical reasoning, and (2) the separation of prefill and decoding phase compression allows for more flexible and task-appropriate token retention strategies.

Table 5.7. KV cache compression methods' performance on Many-shot GSM8K

Method	100%	40%	30%	20%	10%
FullKV	0.8235	-	-	-	-
StreamingLLM	-	0.8037	0.7835	0.7537	0.7432
H2O	-	0.7832	0.7932	0.7428	0.5127
PyramidKV	-	0.7834	0.7934	0.7832	0.7037
SnapKV	-	0.7935	0.8038	0.7934	0.6827
ChunkKV	-	0.7832	0.7932	0.7835	0.7932
ShotKV(Ours)	-	**0.8107**	**0.8082**	**0.8057**	**0.8037**

Conclusion This paper presents a systematic study of KV cache compression's impact on LLMs' core capabilities, revealing several key findings: (1) Performance degradation is highly task-dependent, with arithmetic reasoning tasks showing particular sensitivity to aggressive compression; (2) Instruct-tuned models demonstrate higher sensitivity to compression compared to their base counterparts; (3) Shorter prompts are more vulnerable to

compression effects; (4) Chunk-level compression strategies show superior performance on complex long-context reasoning tasks; (5) Long-context generation tasks are more sensitive to compression than long-context reasoning tasks.

Based on these insights, we proposed ShotKV, a novel compression method that separately handles prefill and decoding phases while preserving shot-level semantic coherence. Our method demonstrates superior performance on long-context arithmetic reasoning tasks and long-context generation tasks, maintaining high accuracy even at low compression ratios.

These findings have important implications for the deployment of LLMs in resource-constrained environments and suggest several promising directions for future research, including: (1) Development of task-adaptive compression strategies; (2) Investigation of compression-aware training methods; and (3) Extension of compression techniques to other model architectures and modalities.

CHAPTER 6

CONCLUSION & FUTURE WORK

6.1 Conclusion

This dissertation has investigated the critical challenge of efficient long-context processing in Large Language Models (LLMs), focusing on KV Cache optimization techniques. As LLMs continue to advance in capability and deployment scope, managing their computational and memory requirements becomes increasingly important for practical applications. Our research has made several significant contributions to address these challenges:

First, we introduced **LongGenBench**, a novel evaluation framework specifically designed to assess LLMs' ability to generate coherent long-form content—a capability distinct from mere information retrieval in long contexts. Our experiments revealed that current models experience substantial performance degradation (ranging from 1.2% to 47.1%) when required to maintain coherence across extended generations, with degradation patterns varying significantly across model architectures and families. These findings highlight a critical gap in existing evaluation methodologies and provide a more comprehensive assessment of long-context processing capabilities.

Second, we proposed **ChunkKV**, a semantic-aware KV Cache compression framework that treats semantically coherent token groups as fundamental compression units rather than isolated tokens. By preserving these semantic chunks holistically, ChunkKV maintains crucial contextual relationships that would otherwise be fragmented through token-level compression. Our extensive evaluations demonstrated that ChunkKV achieves up to 10% performance improvement under aggressive compression scenarios compared to state-of-the-art methods across diverse benchmarks, including challenging reasoning tasks and long-context assessments.

Third, our **comprehensive empirical study** of KV Cache compression's impact on LLMs' fundamental abilities revealed several important insights: (1) performance degradation is highly task-dependent, with arithmetic reasoning tasks showing particular sensitivity; (2) instruct-tuned models demonstrate higher compression sensitivity compared to their base counterparts; (3) chunk-level compression strategies show superior performance on complex long-context reasoning tasks; and (4) long-context generation tasks are more

vulnerable to compression effects than retrieval-focused long-context tasks. Based on these findings, we developed **ShotKV**, a novel compression method that separately handles prefill and decoding phases while preserving shot-level semantic coherence. This approach has demonstrated superior performance on long-context arithmetic reasoning tasks, maintaining high accuracy even at low compression ratios.

Collectively, these contributions advance our understanding of efficient long-context processing in LLMs and provide practical techniques for optimizing resource utilization while maintaining model performance. Our work highlights the importance of considering task-specific requirements when designing compression strategies and demonstrates that semantic-aware approaches can significantly outperform methods that treat tokens as independent units.

6.2 Future Work

While our research has made significant strides in efficient long-context processing, several promising directions remain for future exploration:

Task-Adaptive Compression Strategies Our findings on task-dependent compression sensitivity suggest the potential for developing adaptive compression methods that dynamically adjust their strategies based on task requirements. Future research could explore frameworks that automatically detect task types from input prompts and apply optimal compression configurations accordingly. This could involve developing meta-learning approaches that learn to predict ideal compression parameters for different task categories.

Compression-Aware Training Current KV Cache compression methods are primarily applied post-training during inference. Integrating compression awareness directly into the training process could potentially yield models inherently more robust to compression effects. This might involve techniques such as compression-simulated training, where models are periodically exposed to compressed KV Caches during the training process, potentially improving their resilience to information loss.

Multimodal Context Compression As multimodal LLMs gain prominence, extending KV Cache optimization techniques to handle diverse data types becomes increasingly important. Future work could explore compression strategies specifically designed for multimodal contexts, addressing the unique challenges of preserving cross-modal relationships while reducing memory requirements.

Enhanced Long-Context Generation Our LongGenBench results revealed significant performance degradation in long-form generation tasks. Future research could focus on architectural modifications or specialized training techniques to enhance models' ability to maintain coherence and consistency across extended generations, potentially through improved attention mechanisms or memory-augmented architectures.

Hardware-Optimized Compression Techniques Exploring compression methods specifically designed to leverage emerging hardware architectures could yield additional efficiency gains. This might involve developing compression strategies optimized for specific accelerators or exploring custom hardware designs tailored to efficient KV Cache management.

In conclusion, efficient long-context processing remains a critical challenge for advancing LLM capabilities while ensuring practical deployability. By addressing the research directions outlined above, future work can build upon our contributions to develop more resource-efficient, capable, and reliable language models for a wide range of applications.

Bibliography

[1] M. Adnan, A. Arunkumar, G. Jain, P. Nair, I. Soloveychik, and P. Kamath, "Keyformer: Kv cache reduction through key tokens selection for efficient generative inference," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 114–127, 2024.

[2] R. Agarwal, A. Singh, L. M. Zhang, B. Bohnet, L. Rosias, S. Chan, B. Zhang, A. Anand, Z. Abbas, A. Nova *et al.*, "Many-shot in-context learning," *arXiv preprint arXiv:2404.11018*, 2024.

[3] C. An, S. Gong, M. Zhong, X. Zhao, M. Li, J. Zhang, L. Kong, and X. Qiu, "L-eval: Instituting standardized evaluation for long context language models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 14 388–14 411.

[4] 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," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 3119–3137.

[5] Y. Bai, S. Tu, J. Zhang, H. Peng, X. Wang, X. Lv, S. Cao, J. Xu, L. Hou, Y. Dong *et al.*, "Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks," *arXiv preprint arXiv:2412.15204*, 2024.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[7] Z. Cai, Y. Zhang, B. Gao, Y. Liu, T. Liu, K. Lu, W. Xiong, Y. Dong, B. Chang, J. Hu *et al.*, "Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling," *arXiv preprint arXiv:2406.02069*, 2024.

[8] P. Cameron R. Wolfe, "Decoder-Only Transformers: The Workhorse of Generative LLMs — cameronrwolfe.substack.com," https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse, [Accessed 28-02-2025].

[9] G. Chen, H. Shi, J. Li, Y. Gao, X. Ren, Y. Chen, X. Jiang, Z. Li, W. Liu, and C. Huang, "Sepllm: Accelerate large language models by compressing one segment into one separator," *arXiv preprint arXiv:2412.12094*, 2024.

[10] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[11] Y. Chen, G. Wang, J. Shang, S. Cui, Z. Zhang, T. Liu, S. Wang, Y. Sun, D. Yu, and H. Wu, "Nacl: A general and effective kv cache eviction framework for llm at inference time," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 7913–7926.

[12] W.-L. Chiang, L. Zheng, Y. Sheng, A. N. Angelopoulos, T. Li, D. Li, B. Zhu, H. Zhang, M. Jordan, J. E. Gonzalez *et al.*, "Chatbot arena: An open platform for evaluating llms by human preference," in *Forty-first International Conference on Machine Learning*, 2024.

[13] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training verifiers to solve math word problems," 2021. [Online]. Available: https://arxiv.org/abs/2110.14168

[14] Z. Dong, T. Tang, J. Li, W. X. Zhao, and J.-R. Wen, "Bamboo: A comprehensive benchmark for evaluating long text modeling capacities of large language models," in *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, 2024, pp. 2086–2099.

[15] 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 preprint arXiv:2407.21783*, 2024.

[16] Y. Feng, J. Lv, Y. Cao, X. Xie, and S. K. Zhou, "Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference," *arXiv preprint arXiv:2407.11550*, 2024.

[17] ——, "Identify critical kv cache in llm inference from an output perturbation perspective," *arXiv preprint arXiv:2502.03805*, 2025.

[18] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020.

[19] Z. Fountas, M. Benfeghoul, A. Oomerjee, F. Christopoulou, G. Lampouras, H. B. Ammar, and J. Wang, "Human-like episodic memory for infinite context llms," in *The Thirteenth International Conference on Learning Representations*.

[20] Y. Fu, Z. Cai, A. Asi, W. Xiong, Y. Dong, and W. Xiao, "Not all heads matter: A head-level kv cache compression method with integrated retrieval and reasoning," *arXiv preprint arXiv:2410.19258*, 2024.

[21] S. Ge, Y. Zhang, L. Liu, M. Zhang, J. Han, and J. Gao, "Model tells you what to discard: Adaptive kv cache compression for llms," in *The Twelfth International Conference on Learning Representations*.

[22] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[23] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *arXiv preprint arXiv:2009.03300*, 2020.

[24] C. Hooper, S. Kim, H. Mohammadzadeh, M. Maheswaran, J. Paik, M. W. Mahoney, K. Keutzer, and A. Gholami, "Squeezed attention: Accelerating long context length llm inference," *arXiv preprint arXiv:2411.09688*, 2024.

[25] C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia, and B. Ginsburg, "Ruler: What's the real context size of your long-context language models?" in *First Conference on Language Modeling*.

[26] A. Jaech, A. Kalai, A. Lerer, A. Richardson, A. El-Kishky, A. Low, A. Helyar, A. Madry, A. Beutel, A. Carney *et al.*, "Openai o1 system card," *arXiv preprint arXiv:2412.16720*, 2024.

[27] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *arXiv preprint arXiv:2406.00515*, 2024.

[28] Y. Jiang, Y. Shao, D. Ma, S. J. Semnani, and M. S. Lam, "Into the unknown unknowns: Engaged human learning through participation in language model agent conversations," *arXiv preprint arXiv:2408.15232*, 2024.

[29] G. Kamradt, "Needle In A Haystack - pressure testing LLMs," *Github*, 2023. [Online]. Available: https://github.com/gkamradt/LLMTest_NeedleInAHaystack/tree/main

[30] T. Kočiskỳ, J. Schwarz, P. Blunsom, C. Dyer, K. M. Hermann, G. Melis, and E. Grefenstette, "The narrativeqa reading comprehension challenge," *Transactions of the Association for Computational Linguistics*, vol. 6, pp. 317–328, 2018.

[31] J. Li, M. Wang, Z. Zheng, and M. Zhang, "Loogle: Can long-context language models understand long contexts?" *arXiv preprint arXiv:2311.04939*, 2023.

[32] 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, 2025.

[33] D. Liu, M. Chen, B. Lu, H. Jiang, Z. Han, Q. Zhang, Q. Chen, C. Zhang, B. Ding, K. Zhang *et al.*, "Retrievalattention: Accelerating long-context llm inference via vector retrieval," *arXiv preprint arXiv:2409.10516*, 2024.

[34] G. Liu, C. Li, J. Zhao, C. Zhang, and M. Guo, "Clusterkv: Manipulating llm kv cache in semantic space for recallable compression," *arXiv preprint arXiv:2412.03213*, 2024.

[35] X. Liu, P. Dong, X. Hu, and X. Chu, "LongGenBench: Long-context generation benchmark," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 865–883. [Online]. Available: https://aclanthology.org/2024.findings-emnlp.48

[36] X. Liu, Z. Tang, H. Chen, P. Dong, Z. Li, X. Zhou, B. Li, X. Hu, and X. Chu, "Can llms maintain fundamental abilities under kv cache compression?" *arXiv preprint arXiv:2502.01941*, 2025.

[37] X. Liu, Z. Tang, P. Dong, Z. Li, B. Li, X. Hu, and X. Chu, "Chunkkv: Semantic-preserving kv cache compression for efficient long-context llm inference," *arXiv preprint arXiv:2502.00299*, 2025.

[38] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyrillidis, and A. Shrivastava, "Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time," *Advances in Neural Information Processing Systems*, vol. 36, pp. 52 342–52 364, 2023.

[39] C. Luo, Y. Shen, Z. Zhu, Q. Zheng, Z. Yu, and C. Yao, "Layoutllm: Layout instruction tuning with large language models for document understanding," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2024, pp. 15 630–15 640.

[40] W. Luo, S. Ma, X. Liu, X. Guo, and C. Xiao, "Jailbreakv: A benchmark for assessing the robustness of multimodal large language models against jailbreak attacks," in *First Conference on Language Modeling*.

[41] Y. Ma, Y. Zang, L. Chen, M. Chen, Y. Jiao, X. Li, X. Lu, Z. Liu, Y. Ma, X. Dong *et al.*, "Mmlongbench-doc: Benchmarking long-context document understanding with visualizations," *Advances in Neural Information Processing Systems*, vol. 37, pp. 95 963–96 010, 2025.

[42] A. Mohtashami and M. Jaggi, "Landmark attention: Random-access infinite context length for transformers," 2023. [Online]. Available: https://arxiv.org/abs/2305.16300

[43] X. Ni, H. Cai, X. Wei, S. Wang, D. Yin, and P. Li, "Xl ^{2} bench: A benchmark for extremely long context understanding with long-range dependencies," *arXiv preprint arXiv:2404.05446*, 2024.

[44] R. Qin, Z. Li, W. He, M. Zhang, Y. Wu, W. Zheng, and X. Xu, "Mooncake: A kvcache-centric disaggregated architecture for llm serving," *arXiv preprint arXiv:2407.00079*, 2024.

[45] K. I. Roumeliotis and N. D. Tselikas, "Chatgpt and open-ai models: A preliminary review," *Future Internet*, vol. 15, no. 6, 2023. [Online]. Available: https://www.mdpi.com/1999-5903/15/6/192

[46] ——, "Chatgpt and open-ai models: A preliminary review," *Future Internet*, vol. 15, no. 6, p. 192, 2023.

[47] U. Shaham, M. Ivgi, A. Efrat, J. Berant, and O. Levy, "Zeroscrolls: A zero-shot benchmark for long text understanding," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 7977–7989.

[48] U. Shaham, E. Segal, M. Ivgi, A. Efrat, O. Yoran, A. Haviv, A. Gupta, W. Xiong, M. Geva, J. Berant *et al.*, "Scrolls: Standardized comparison over long language sequences," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 12 007–12 021.

[49] Y. Shao, Y. Jiang, T. A. Kanell, P. Xu, O. Khattab, and M. S. Lam, "Assisting in writing wikipedia-like articles from scratch with large language models," *arXiv preprint arXiv:2402.14207*, 2024.

[50] A. Talmor, J. Herzig, N. Lourie, and J. Berant, "CommonsenseQA: A question answering challenge targeting commonsense knowledge," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*,

J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4149–4158. [Online]. Available: https://aclanthology.org/N19-1421/

[51] H. Tang, Y. Lin, J. Lin, Q. Han, S. Hong, Y. Yao, and G. Wang, "Razorattention: Efficient kv cache compression through retrieval heads," *arXiv preprint arXiv:2407.15891*, 2024.

[52] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han, "Quest: query-aware sparsity for efficient long-context llm inference," in *Proceedings of the 41st International Conference on Machine Learning*, 2024, pp. 47 901–47 911.

[53] G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[54] A. Wang, H. Chen, J. Tan, K. Zhang, X. Cai, Z. Lin, J. Han, and G. Ding, "Prefixkv: Adaptive prefix kv cache is what vision instruction-following models need for efficient generation," *arXiv preprint arXiv:2412.03409*, 2024.

[55] X. Wang, M. Salmani, P. Omidi, X. Ren, M. Rezagholizadeh, and A. Eshaghi, "Beyond the limits: a survey of techniques to extend the context length in large language models," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 2024, pp. 8299–8307.

[56] T. Wu, S. He, J. Liu, S. Sun, K. Liu, Q.-L. Han, and Y. Tang, "A brief overview of chatgpt: The history, status quo and potential future development," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 5, pp. 1122–1136, 2023.

[57] C. Xiao, P. Zhang, X. Han, G. Xiao, Y. Lin, Z. Zhang, Z. Liu, and M. Sun, "Infllm: Training-free long-context extrapolation for llms with an efficient context memory," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[58] G. Xiao, J. Tang, J. Zuo, J. Guo, S. Yang, H. Tang, Y. Fu, and S. Han, "Duoattention: Efficient long-context llm inference with retrieval and streaming heads," *arXiv preprint arXiv:2410.10819*, 2024.

[59] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," in *The Twelfth International Conference on Learning Representations*.

[60] D. Yang, X. Han, Y. Gao, Y. Hu, S. Zhang, and H. Zhao, "Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference," in *Findings of the Association for Computational Linguistics ACL 2024*, 2024, pp. 3258–3270.

[61] Z. Zeng, B. Lin, T. Hou, H. Zhang, and Z. Deng, "In-context kv-cache eviction for llms via attention-gate," *arXiv preprint arXiv:2410.12876*, 2024.

[62] P. Zhang, Z. Liu, S. Xiao, N. Shao, Q. Ye, and Z. Dou, "Long context compression with activation beacon," in *The Thirteenth International Conference on Learning Representations*.

[63] X. Zhang, Y. Chen, S. Hu, Z. Xu, J. Chen, M. Hao, X. Han, Z. Thai, S. Wang, Z. Liu *et al.*, "∞ bench: Extending long context evaluation beyond 100k tokens," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 15 262–15 277.

[64] X. Zhang, C. Du, C. Du, T. Pang, W. Gao, and M. Lin, "Simlayerkv: A simple framework for layer-level kv cache reduction," *arXiv preprint arXiv:2410.13846*, 2024.

[65] Y. Zhang, Y. Hu, R. Zhao, J. Lui, and H. Chen, "Unifying kv cache compression for large language models with leankv," *arXiv preprint arXiv:2412.03131*, 2024.

[66] 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.

[67] J. Zhao, Z. Fang, S. Li, S. Yang, and S. He, "Buzz: Beehive-structured sparse kv cache with segmented heavy hitters for efficient llm inference," *arXiv preprint arXiv:2410.23079*, 2024.

[68] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *Advances in Neural Information Processing Systems*, vol. 36, pp. 46 595–46 623, 2023.

[69] X. Zhou, W. Wang, M. Zeng, J. Guo, X. Liu, L. Shen, M. Zhang, and L. Ding, "Dynamickv: Task-aware adaptive kv cache compression for long context llms," *arXiv preprint arXiv:2412.14838*, 2024.