CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Adaptive KV Cache Management for Efficient Transformer-based LLM Inference

Leveraging Attention Sparsity for Memory Optimization

Master's thesis in Computer science and engineering

DIKAI XU

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Adaptive KV Cache Management for Efficient Transformer-based LLM Inference

Leveraging Attention Sparsity for Memory Optimization

DIKAI XU

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Adaptive KV Cache Management for Efficient Transformer-based LLM Inference
Leveraging Attention Sparsity for Memory Optimization
DIKAI XU

Supervisor: Ahmed Ali-Eldin Hassan, Department of Computer Science and Engineering
Examiner: Ahmed Ali-Eldin Hassan, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Description of the picture on the cover page (if applicable)

Typeset in LaTeX
Gothenburg, Sweden 2025

Adaptive KV Cache Management for Efficient Transformer-based LLM Inference
Leveraging Attention Sparsity for Memory Optimization
DIKAI XU
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This Master's thesis addresses the critical challenge of memory inefficiency in Transformer-based Large Language Models (LLMs) during inference, specifically focusing on the prohibitive memory footprint of the Key-Value (KV) cache. As LLMs scale, the KV cache becomes a significant bottleneck, limiting longer context windows and overall operational efficiency. To mitigate this issue, we propose and evaluate Adap-KV, a novel adaptive memory management strategy for the KV cache. Adap-KV employs a layer-aware dynamic allocation approach that intelligently adjusts KV cache size in real-time, leveraging insights from attention sparsity patterns. Our method aims to optimize memory utilization without compromising the performance or quality of LLM inference. Experimental results demonstrate that Adap-KV significantly reduces KV cache memory consumption, thereby enhancing the efficiency and scalability of Transformer-based LLMs, making them more amenable for real-world deployments with extended context capabilities.

Keywords: Large Language Models, Transformers, KV Cache, Memory Optimization, Adaptive Memory Management, Attention Sparsity, Deep Learning Inference, Resource Efficiency

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor Ahmed Ali-Eldin Hassan, for his invaluable guidance, continuous support, and insightful feedback throughout this master's thesis project. His expertise in the field and dedication to my learning have been instrumental in the completion of this work.

I would also like to thank the Department of Computer Science and Engineering at Chalmers University of Technology for providing the resources and environment necessary for this research.

Finally, I extend my deepest appreciation to my family and friends for their unwavering encouragement and understanding during my studies. Their support has been a constant source of motivation.

Dikai Xu, Gothenburg, 2025-06-13

Contents

List of Figures

List of Tables

1

Introduction

Transformer-based large language models (LLMs), exemplified by prominent models like GPT, BERT, LLaMA, and similar architectures, have significantly reshaped the landscape of natural language processing. Their powerful capabilities facilitate complex applications ranging from automated customer service, advanced content generation, real-time language translation, sophisticated data analytics, to human-like conversational agents. Despite their widespread adoption and impressive versatility, the rapid evolution and deployment of these models are increasingly hampered by substantial memory demands. These memory constraints present critical challenges, especially within hardware-limited contexts, including edge devices, mobile applications, consumer-grade graphics processing units (GPUs), and cost-sensitive cloud environments.

At the core of Transformer-based models lies the attention mechanism, a revolutionary innovation enabling models to dynamically adjust focus across diverse segments of input sequences. This adaptability significantly improves performance on complex tasks requiring nuanced contextual understanding and extensive relational analysis over long data sequences. However, the advantage of attention mechanisms incurs significant computational and memory overheads. Specifically, Transformers must store extensive intermediate computational results, known as key-value (KV) caches, during inference. KV caches retain representations of previously processed tokens, enabling dynamic computation of attention scores at each inference iteration. Consequently, their memory footprint grows proportionally with sequence length, imposing severe constraints on resource utilization and efficiency.

1.1 Memory Challenges in Transformer Models

Memory usage during Transformer inference broadly encompasses three primary areas: model weights, intermediate activations, and the dynamically expanding key-value cache. Model weights constitute the trained parameters, typically having fixed and predictable memory requirements. Intermediate activations represent temporary computational results, also predictable to a certain extent and relatively amenable to existing optimization strategies such as weight quantization, precision reduction, and sparsification. However, the dynamic and unpredictable nature of KV cache memory introduces unique challenges.

The KV cache dynamically grows during autoregressive inference as the model se-

quentially generates new tokens, linearly expanding with each step. Thus, memory demand quickly escalates as the inference progresses, particularly when handling extensive contexts spanning thousands of tokens. For instance, a Transformer model containing billions of parameters, performing inference over thousands of tokens, can easily require tens of gigabytes of dedicated GPU memory solely for KV cache storage. Such expansive memory requirements significantly curtail inference throughput, restrict feasible batch sizes, limit the scalability of deployments, and raise hardware and operational costs. As applications increasingly demand longer input sequences to enhance the coherence, accuracy, and relevance of generated outputs, managing this memory overhead becomes critically urgent.

1.2 KV Cache as a Bottleneck

The KV cache represents an exceptionally challenging component of memory management due to its dynamic growth and inherently context-dependent nature. Each inference step in Transformers produces and stores new token representations for every layer, which are recurrently referenced to calculate attention scores. Traditional implementation strategies often employ static memory allocation for a predetermined maximum context size. While this static approach simplifies memory handling, it is notoriously inefficient, frequently allocating substantially more memory than actually required by real-time inference tasks.

Emerging research has highlighted pronounced sparsity within Transformer attention patterns, revealing that typically only a limited subset of cached tokens significantly influences inference results. Nonetheless, conventional implementations continue to store extensive full-context representations for every token. This inefficiency leads to substantial wastage of memory resources, exacerbating hardware constraints in resource-limited contexts. Particularly in mobile devices, embedded systems, edge computing scenarios, and economically constrained cloud deployments, these inefficiencies pose significant operational and economic challenges.

1.3 Addressing Memory Bottlenecks

Memory bottlenecks have cascading effects on the practicality and efficiency of deploying LLMs:

- Limited Batch Size: To mitigate memory pressure, developers often resort to smaller inference batch sizes. While this reduces peak memory usage, it simultaneously underutilizes the parallel processing capabilities of modern accelerators, leading to lower throughput and longer inference times. This is particularly problematic for real-time applications requiring low latency.
- Reduced Context Length: The rapidly growing KV cache directly restricts the maximum context length that an LLM can handle. This is a critical limitation for applications requiring extensive understanding of long documents, conversations, or code, such as summarization of lengthy articles, long-form

question answering, or complex code generation. When the context exceeds the memory capacity, the model must either truncate the input or resort to less efficient techniques like sliding windows, which can degrade performance.

- Increased Hardware Requirements and Costs: To overcome memory constraints, powerful and expensive hardware with vast amounts of VRAM (Video RAM) is often required. This significantly increases the operational costs associated with deploying and maintaining LLM services, making them less accessible for smaller organizations or researchers.
- Energy Consumption: Higher memory usage and the need for more powerful hardware also translate to increased energy consumption, contributing to the environmental footprint of large-scale AI deployments.

Given these challenges, effective memory management strategies are paramount for unlocking the full potential of Transformer-based LLMs. Optimizing memory usage during inference is not merely about enabling larger models or longer contexts; it is about making LLMs more accessible, efficient, and sustainable.

To mitigate memory-related bottlenecks, researchers and practitioners have explored diverse optimization methods. Techniques such as model quantization and precision scaling effectively reduce memory consumption by compressing static weights but typically fail to address dynamically expanding structures such as the KV cache adequately. Sparse representations aim to minimize storage requirements by pruning less critical weights but similarly neglect the adaptive management of dynamically fluctuating inference workloads. Innovative solutions, including methods like PagedAttention, have introduced dynamic memory allocation by partitioning KV cache memory into manageable fixed-size blocks allocated as necessary, significantly reducing wastage and enhancing utilization.

However, these existing methods typically distribute cache memory uniformly across model layers, disregarding empirical observations of layer-specific attention usage patterns. Consequently, a more nuanced, adaptive, and layer-aware approach to KV cache management emerges as necessary to achieve optimal memory efficiency.

Dynamic layer-aware cache allocation presents a promising avenue to resolve these inefficiencies effectively. Leveraging real-time attention metrics–such as the Gini coefficient, attention entropy, or other statistical indicators–this approach dynamically adapts cache allocations according to actual usage patterns. Such adaptive strategies significantly reduce memory demands without adversely affecting inference accuracy or latency, providing superior scalability with context length compared to traditional static methods.

1.4 Objectives of This Thesis

This thesis explicitly targets optimizing memory usage in Transformer-based LLM inference with a special focus on developing adaptive KV cache management strategies. The core objective is:

Developing Adaptive KV Cache Management: Propose, design, and implement a novel adaptive memory allocation method that dynamically adjusts KV cache sizes based on real-time attention sparsity metrics such as Gini coefficients and attention entropy, enabling more efficient memory utilization.

By focusing explicitly on this objective, this thesis aims to rigorously and innovatively address KV cache-related memory inefficiencies, thereby facilitating improved scalability, reduced hardware requirements, and broader accessibility for Transformer-based language models.

1.5 Thesis Contributions

This research contributes a thorough and detailed analysis of contemporary Transformer memory usage, with a targeted emphasis on the dynamically scaling KV cache. Specifically, the thesis proposes and rigorously evaluates a novel adaptive memory management strategy. This strategy dynamically allocates KV cache resources based on real-time analyses of attention distribution patterns across Transformer layers. Through comprehensive experimental evaluations conducted on benchmark datasets like LoogGle and Rluer, the proposed method demonstrates significant reductions in memory usage with minimal impacts on inference performance and accuracy.

In conclusion, this thesis provides critical advancements in memory optimization methodologies essential for the practical, efficient, and scalable deployment of Transformer-based language models. The outcomes of this research enable broader accessibility, facilitate cost-effective deployments, and significantly enhance the operational efficiency of large-scale language models across diverse computational environments.

2

Background and Related Work

The landscape of artificial intelligence has been profoundly reshaped by LLMs, which demonstrate exceptional abilities in comprehending nuanced context and executing logical reasoning tasks. These models, often trained on vast datasets, have found applications across natural language processing, computer vision, and beyond. Architectures like GPT and LLaMA, which are at the forefront of this revolution, are predominantly built upon the Transformer framework. A critical element for enhancing the operational speed of these models, especially during the text generation phase, involves sophisticated management of Key-Value (KV) caches. A foundational understanding of the Transformer's mechanics and the specific function of the KV cache is therefore crucial before delving into optimization techniques.

2.1 Transformer Architecture

Transformers have become the dominant architecture for LLMs due to their unparalleled capacity to model long-range dependencies in sequential data like text. This capability underpins their success in tasks ranging from translation to generation. While the original Transformer included both an encoder and a decoder, many modern LLMs employ only the decoder component. Our discussion will center on the decoder's essential elements pertinent to KV cache mechanisms, omitting details like layer normalization for clarity.

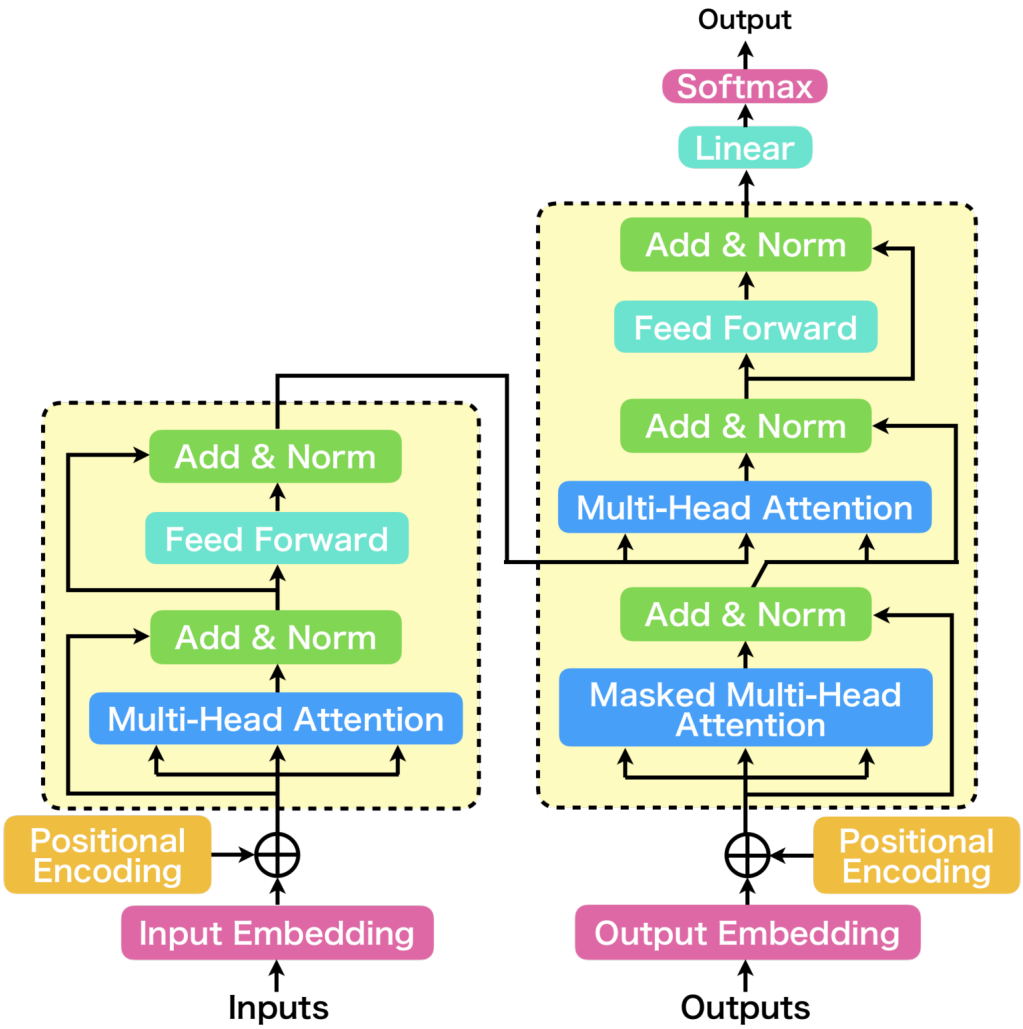

Figure 2.1: Transformer Architecture

2.1.1 Transformer block

A Transformer decoder typically comprises a stack of identical layers, often referred to as Transformer blocks. Each block sequentially refines the representation of the input sequence. Within every block, two main sub-layers operate: a Multi-Head Self-Attention (MHSA) unit and a Position-wise Feed-Forward Network (FFN).

Input Representation and Positional Context: The process begins by converting an input text sequence X into a sequence of tokens, $X = [x_1, x_2, \ldots, x_{|X|}]$. An embedding layer then transforms these discrete tokens into continuous vector representations, yielding a matrix $\mathbf{X} \in \mathbb{R}^{|X| \times d_x}$, where d_x is the model's embedding dimension. Since the self-attention mechanism itself does not inherently process sequence order, positional information must be added. This is achieved by incorporating positional encodings, $PE(X) \in \mathbb{R}^{|X| \times d_x}$, typically using functions based on sine and cosine waves, into the token embeddings. The resulting sequence representation, $\mathbf{X}' = \mathbf{X} + PE(X)$, serves as the input to the first Transformer block.

Multi-Head Self-Attention (MHSA): MHSA enables the model to dynamically

weigh the significance of different tokens when computing the representation for any given token. For an input $\mathbf{X}' \in \mathbb{R}^{|X| \times d_x}$ entering a Transformer block, MHSA first generates multiple projections for each token, creating sets of Query (Q_i), Key (K_i), and Value (V_i) vectors per attention head i. These are calculated via learned linear transformations:

$$Q_i = \mathbf{X}'W_{Q_i}, \quad K_i = \mathbf{X}'W_{K_i}, \quad V_i = \mathbf{X}'W_{V_i}$$

where $W_{Q_i}, W_{K_i} \in \mathbb{R}^{d_x \times d_k}$ and $W_{V_i} \in \mathbb{R}^{d_x \times d_v}$ are trainable weight matrices unique to head i. d_k denotes the dimension for queries and keys, while d_v is the dimension for values.

The attention scores are computed using the scaled dot-product attention function for each head:

$$\text{Attention}(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}}\right) V_i$$

The softmax output provides weights determining how strongly each query token attends to each key token (representing all tokens in the sequence). The division by $\sqrt{d_k}$ is a scaling factor crucial for maintaining stable training dynamics.

The outputs Z_i from all h attention heads are then combined. They are first concatenated along the feature dimension and subsequently passed through a final linear projection, defined by the weight matrix $W_O \in \mathbb{R}^{hd_v \times d_x}$:

$$Z = \text{Concat}(Z_1, Z_2, \ldots, Z_h)W_O$$

This aggregation step allows the model to jointly attend to information from different representational subspaces identified by the various heads.

Feed-Forward Network (FFN): Following the MHSA layer (usually accompanied by residual connections and layer normalization, omitted here), the resulting tensor Z is processed by a position-wise FFN. This sub-layer applies the same transformation independently to each token's representation. It typically comprises two linear layers separated by a non-linear activation function σ (such as ReLU or GeLU):

$$\text{FFN}(Z) = \sigma(ZW_1 + b_1)W_2 + b_2$$

With weight matrices $W_1 \in \mathbb{R}^{d_x \times d_{ff}}$ and $W_2 \in \mathbb{R}^{d_{ff} \times d_x}$, and bias vectors $b_1 \in \mathbb{R}^{d_{ff}}$ and $b_2 \in \mathbb{R}^{d_x}$. The intermediate dimension d_{ff} is often considerably larger than the model dimension d_x. This network provides additional modeling capacity, transforming the attention outputs before they are passed to the subsequent Transformer block or used for the final prediction.

2.2 The Auto-Regressive Generation Process

Text generation in LLMs commonly follows an auto-regressive paradigm, producing the output sequence one token after another. At any given step t, the model aims to predict the subsequent token, x_{t+1}, based on the sequence generated so

far, $X_{[1:t]} = [x_1, x_2, \ldots, x_t]$. This involves calculating the conditional probability distribution $P(x_{t+1}|x_1, \ldots, x_t)$. Operationally, the model uses the final hidden state representation h_t (corresponding to the t-th token) and projects it through an output layer (parameterized by W_{out} and b_{out}) to obtain scores (logits) for each potential next token in the vocabulary. A softmax function then converts these logits into probabilities:

$$P(x_{t+1}|x_1, \ldots, x_t) = \text{softmax}(h_t W_{out} + b_{out})$$

The actual next token x_{t+1} is typically chosen by sampling from this distribution. This new token is then added to the sequence, extending it to $X_{[1:t+1]}$, and the generation cycle continues until a predefined stopping criterion is met, like generating a special end-of-sequence token or reaching a maximum length constraint.

2.3 Optimizing Generation with the Key-Value (KV) Cache

The auto-regressive nature of generation, while effective, introduces a computational inefficiency. When generating the $(t+0)$-th token, the standard attention mechanism recalculates attention scores over the entire sequence processed thus far, $X_{[1:t+1]}$. This necessitates recomputing the Key (K) and Value (V) matrices derived from tokens x_1 through x_t at every single generation step. This redundant computation causes the inference time to grow quadratically with the sequence length, becoming a bottleneck for long outputs.

The Key-Value (KV) cache optimization addresses this redundancy directly. It works by storing the Key and Value vectors computed for tokens in previous steps and reusing them, avoiding repeated calculations.

2.3.1 How KV Caching Works

Consider the generation at step t. The Key and Value matrices corresponding to the tokens $x_1, \ldots, x_{t-1}$ were already computed during the generation of x_t. The KV cache maintains these previously computed matrices, let's call them $\hat{K}_i^{t-1}$ and $\hat{V}_i^{t-1}$ for attention head i. When generating x_{t+1}, the model only needs to compute the Key vector k_i^t and Value vector v_i^t for the most recent input token x_t. These newly computed vectors are then simply appended to the cached matrices:

$$K_i^t = \text{Concat}(\hat{K}_i^{t-1}, k_i^t), \quad V_i^t = \text{Concat}(\hat{V}_i^{t-1}, v_i^t)$$

Similarly, the Query vector q_i^t is computed only based on x_t. The attention computation for generating the output at step t then uses q_i^t and the extended Key and Value matrices K_i^t and V_i^t:

$$Z_i^t = \text{softmax}\left(\frac{q_i^t (K_i^t)^\top}{\sqrt{d_k}}\right) V_i^t$$

This reuse of cached Keys and Values eliminates the need to re-project and reprocess all prior tokens for their K and V representations at each step, leading to significant speedups, especially as the generated sequence grows longer.

2.3.2 Impact on Computational Resources

KV caching introduces a trade-off: it reduces computation time but increases memory usage.

Computational Savings: The major advantage is the avoidance of redundant computations for Keys, Values, and parts of the attention score calculation for all tokens preceding the current one. The total time saved accumulates with each generated token and is proportional to the number of tokens already cached (t_c) and the sequence length (t), making the generation process scale more linearly rather than quadratically with sequence length.

Memory Costs: The downside is the need for additional memory to store the computed Key and Value vectors for all cached tokens (t_c), across all L layers and all h attention heads per layer. The size of this cache scales linearly with t_c, the number of layers L, the number of heads h, and the dimensions of the Key/Value vectors (d_k, d_v). Typically, for a model dimension d_x, this storage requirement is roughly $O(L \cdot t_c \cdot d_x \cdot \text{bytes\_per\_value})$. For long sequences, this cache can consume gigabytes of memory, often becoming the limiting factor for context length on hardware like GPUs. Figure 2.2 shows the KV Cache growth for the Model Qwen2.5-1.5B with different input length.

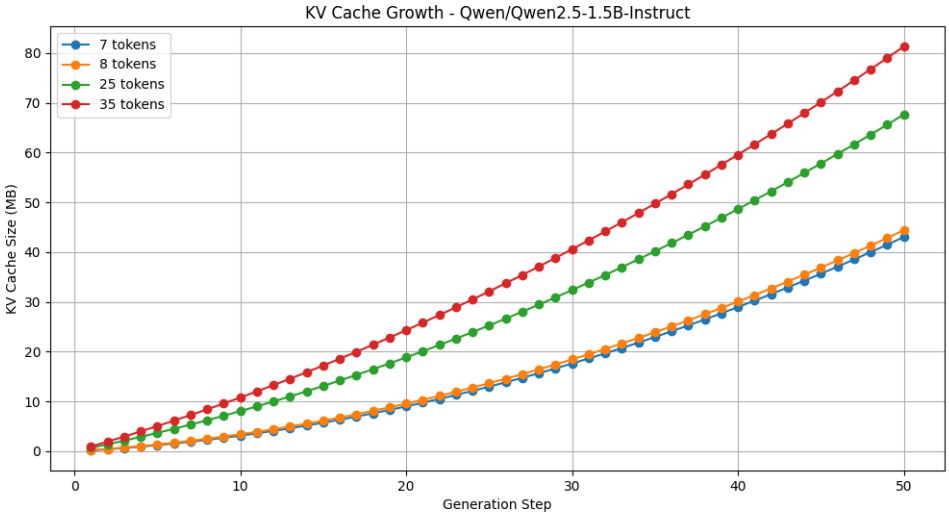

Figure 2.2: KV Cache Growth Example

2.4 Obstacles in KV Cache Utilization

Effectively leveraging the KV cache introduces its own set of difficulties, which become more pronounced with increasing model size and sequence length:

- **Eviction Strategy:** When cache memory is full, deciding which KV pairs to remove is non-trivial. Simple strategies like LRU might discard information crucial for maintaining context or coherence in LLMs, as token importance

isn't always tied to recency. Developing effective eviction policies is an active area of research.

- **Memory Footprint Management:** The substantial memory demand of the KV cache, especially for contexts spanning thousands or tens of thousands of tokens, can exceed the capacity of high-bandwidth GPU memory. Managing this requires careful allocation and potentially offloading strategies to slower memory tiers, which adds system complexity.
- **Latency Concerns:** The process of accessing, modifying, and potentially moving large KV cache tensors at every decoding step can introduce latency, particularly if memory bandwidth is limited or if data needs to be transferred between CPU and GPU memory.
- **Compression vs. Accuracy:** Reducing the cache size via methods like quantization (lowering numerical precision) or structural compression (e.g., low-rank approximation) can save memory but may degrade the model's output quality by introducing errors. Striking an optimal balance is essential but difficult.
- **Adaptability to Dynamic Inputs:** LLM usage often involves varied sequence lengths and interaction patterns. Fixed caching strategies might be inefficient. Adaptive systems that can dynamically adjust cache size, eviction policies, or compression levels based on the current workload are needed but add complexity.
- **Challenges in Distributed Systems:** Scaling inference across multiple devices requires distributing and synchronizing the KV cache. Maintaining consistency, ensuring fault tolerance, and minimizing communication overhead for cache updates present significant system design challenges.

Overcoming these hurdles is crucial for deploying LLMs efficiently in demanding scenarios that require handling extensive contexts or providing rapid responses.

2.5 Related Work

The challenge of managing the KV cache in LLMs to balance inference speed and memory consumption has spurred a variety of optimization techniques. Existing approaches can be broadly classified into three main categories based on the level at which the optimization is applied: strategies operating directly on the cache tokens, modifications to the underlying model architecture, and system-level enhancements coordinating execution and memory.

2.5.1 Token-Focused Optimization Strategies

This class of methods aims to improve KV cache efficiency by manipulating the cached token data itself, without altering the fundamental architecture of the LLM. These techniques focus on the fine-grained selection, organization, and representation of the individual key-value pairs associated with each token.

Key approaches within this category include:

- **KV Cache Selection:** These methods prioritize which tokens' KV pairs are most important to retain in the cache, especially when memory is limited. Strategies range from static selection[1][2] performed once after processing the initial prompt to dynamic selection methods[3][4] that continuously update the cache during generation, sometimes involving permanent eviction of less relevant tokens or offloading them to slower memory.
- **KV Cache Budget Allocation:** Rather than applying uniform compression or eviction, these techniques distribute the available memory budget intelligently across different parts of the cache. Allocation decisions can be made at the granularity of model layers[5][6] or even individual attention heads[7], assigning more resources to components deemed more critical for maintaining accuracy.
- **KV Cache Merging:** To reduce redundancy, merging techniques consolidate similar or related KV pairs. Merging can occur within a single layer (intra-layer) by combining representations of adjacent or semantically similar token[8][9], or across different layers (cross-layer) by identifying and sharing redundant KV information between layers[10][11].
- **KV Cache Quantization:** This popular technique reduces the memory footprint by lowering the numerical precision used to store the key and value vectors (e.g., from 16-bit floating-point to 8-bit or 4-bit integers). Methods vary from applying a fixed precision uniformly[12][12], to using mixed precision where more important elements are stored with higher fidelity[13][14], and techniques specifically designed to handle outlier values that can disproportionately affect quantization quality[15].
- **KV Cache Low-Rank Decomposition:** Leveraging the observation that KV cache matrices often exhibit low-rank structure, these methods compress the cache by representing the keys and values using fewer dimensions. Techniques include applying Singular Value Decomposition (SVD)[16][17], using tensor decomposition methods[18], or employing learned transformations to find efficient low-rank approximations[19][20].

2.5.2 Model Architecture Modifications

This category encompasses approaches that modify the LLM's architecture itself to inherently reduce KV cache requirements or facilitate more efficient cache usage. These methods often require model retraining or fine-tuning.

Prominent strategies include:

- **Attention Grouping and Sharing:** These techniques reduce the total number of unique key and value vectors that need to be computed and stored. This can involve grouping multiple query heads to share a single key/value head within a layer (like Multi-Query or Grouped-Query Attention)[21][22] or sharing key/value heads or even attention weights across multiple layers[23][24].

- **Architecture Alteration:** This involves more fundamental changes to the attention mechanism or the overall model structure. Examples include developing novel attention variants that are inherently less reliant on extensive caching (e.g., using linear attention approximations or compressive memory techniques)[25][26] or augmenting the standard decoder architecture with additional components (like a separate context encoder or cross-attention layers)[27][28] to manage context information more compactly.
- **Non-Transformer Architectures:** Recognizing the inherent scaling limitations of Transformer attention, researchers are exploring alternative sequence modeling architectures inspired by Recurrent Neural Networks (RNNs) or State Space Models (SSMs). Models like RWKV or Mamba, and hybrid approaches combining Transformer elements with these alternatives, aim to achieve strong performance with significantly lower (or constant) memory requirements during inference, effectively bypassing the traditional KV cache bottleneck.

2.5.3 System-Level Optimizations

Complementary to token-level and model-level techniques, system-level optimizations focus on the runtime environment, execution scheduling, and hardware interaction to manage the KV cache efficiently.

Key areas are:

- **Memory Management:** Drawing inspiration from operating systems, these methods implement sophisticated memory allocation and management schemes for the KV cache. Techniques include virtual memory systems with paging (allowing non-contiguous storage and reducing fragmentation), efficient memory pools, and prefix-aware designs that detect and share common KV cache prefixes among different requests in a batch.
- **Scheduling:** The order in which inference requests are processed can significantly impact cache utilization and overall throughput. Scheduling strategies include prefix-aware scheduling (grouping requests with shared context), preemptive scheduling (allowing high-priority requests to interrupt lower-priority ones, often involving swapping cache states), fairness-oriented scheduling (balancing throughput and latency across users), and layer-specific or hierarchical scheduling (optimizing cache access based on layer-specific needs or managing cache across different memory tiers).
- **Hardware-Aware Design:** These optimizations tailor KV cache management strategies to the specific characteristics of the underlying hardware. This includes optimizing GPU kernel implementations for attention computation and memory access patterns, designing efficient data transfer strategies for I/O-bound scenarios (e.g., between GPU HBM and SRAM, or GPU and CPU memory), orchestrating workloads across heterogeneous compute units (CPU-GPU), and leveraging novel storage technologies like SSDs or computational storage for cache offloading or even in-storage processing.

Collectively, these three broad categories represent a rich and rapidly evolving landscape of techniques aimed at making large language model inference more efficient and scalable by tackling the critical bottleneck of Key-Value cache management.

3

Methods

3.1 Motivating Observations and Rationale

The deployment of large-scale Transformer models for tasks involving long sequence inference is significantly constrained by the memory requirements of the Key-Value (KV) cache. This cache, essential for efficient autoregressive generation, stores the key and value vectors from all preceding tokens for every attention layer. As the input or generated sequence length increases, the size of this KV cache, and consequently its memory consumption and the computational cost of attending over it, scales linearly. This often becomes the primary bottleneck, limiting the practical context length and throughput of these powerful models.

A fundamental inefficiency in standard Transformer inference arises from the conventional static allocation of the KV cache. Typically, memory is reserved for a fixed maximum context window, and all key-value pairs within this window are uniformly maintained for each layer. However, empirical observations and analysis of attention mechanisms reveal a crucial insight: not all tokens stored in the KV cache are equally important or actively utilized by the model's attention heads at every generation step. More specifically, preliminary investigations undertaken for this work, supported by patterns observed in existing literature, demonstrate that attention sparsity is markedly heterogeneous across the different layers of a Transformer model.

It has been observed that different layers exhibit distinct attentional behaviors. Some layers might consistently focus their attention on a very small subset of tokenssuch as the initial "attention sink" tokens or specific, highly relevant features within the prompt. In contrast, other layers might require a more distributed attention mechanism, drawing context from a broader range of past tokens. This layer-specific variability in attention focus implies that a significant portion of a uniformly allocated KV cache often remains underutilized by many layers, leading to wasteful memory consumption and unnecessary computations over less relevant cached entries. For instance, if a particular layer at a given timestep primarily attends to the most recent 50 tokens, maintaining a cache of several thousand tokens for that specific layer offers diminishing returns at a high resource cost.

This observed inefficiency of static, one-size-fits-all KV caching provides the primary motivation for the proposed methodology. If the varying degrees of attention

sparsity within each layer can be accurately quantified in real-time during inference, it becomes feasible to dynamically tailor the KV cache allocation for each layer. Such a dynamic approach could significantly reduce overall memory demand by allocating substantial cache only to those layers that demonstrably benefit from a wider context, while aggressively pruning the cache for layers exhibiting highly focused attention. This stands in contrast to simplistic static strategies, such as fixed-window caching or arbitrary eviction of the oldest tokens, which do not consider the actual importance of tokens to specific layers. The overarching goal is to develop a training-free and model-agnostic dynamic KV cache management system that intelligently adapts to these observed sparsity patterns, thereby enabling more efficient long-context inference by minimizing memory overhead and reducing extraneous attention computations, all while striving to maintain the model's original performance.

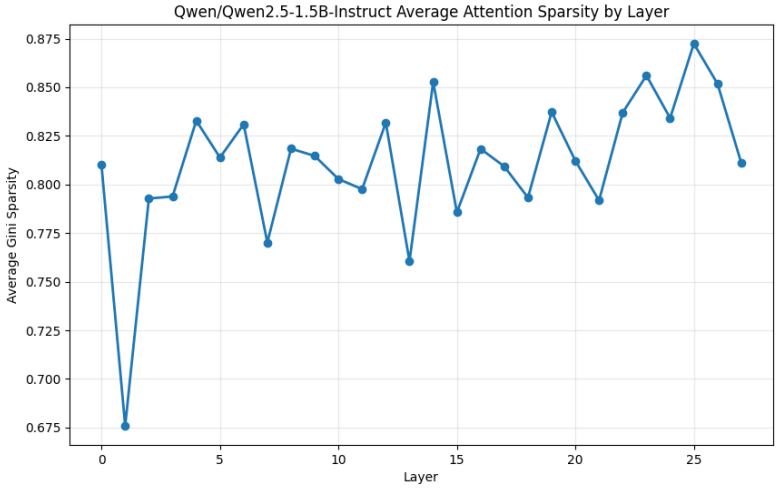

Figure 3.1: Visualization of attention sparsity across several Transformer layers. Many positions exhibit low attention scores, suggesting that only a few key tokens dominate the attention landscape.

3.2 Core Strategy: Real-Time Adaptation based on Attention Sparsity

The proposed method introduces a layer-aware dynamic allocation strategy for KV cache management. The central idea is to adjust the KV cache size for each Transformer layer on-the-fly, directly informed by the sparsity of its attention distribution as observed during the ongoing autoregressive generation process. This allows the system to optimize memory usage and computational workload contextually, without requiring any modifications to the pretrained model weights or subsequent fine-tuning.

3.3 Continuous Monitoring of Per-Layer Attention Dynamics

The foundation of our dynamic allocation mechanism is the continuous monitoring and quantification of attention patterns within each layer of the Transformer model during the generation process. As the model generates tokens one by one, the attention mechanism in each layer computes a distribution of scores indicating the relevance of previously processed tokens (stored in the KV cache) to the current token being generated. Our method intercepts these attention scores at each step.

Rather than treating the KV cache as a monolithic block or assuming uniform contextual needs across all layers, our approach performs a granular, layer-specific analysis. For each layer l in the Transformer stack (where l ranges from 1 to L, the total number of layers), we quantify the characteristics of its attention distribution. Specifically, we calculate the Gini coefficient, denoted as G^l, based on the layer's aggregated attention distribution over the tokens currently held in its portion of the KV cache. The Gini coefficient, a statistical measure often used to describe inequality in distributions, serves here as a robust, real-time indicator of attention concentration:

- A high value of G^l (approaching 1) signifies a highly sparse or concentrated attention pattern for that specific layer l. This implies that the layer is directing most of its attentional weight towards a very small fraction of the available tokens in its KV cache. Consequently, a large portion of the cached information for this layer might be less critical for its immediate computational needs.
- Conversely, a low value of G^l (closer to 0) indicates a more uniform, dense, or diffuse attention distribution. In this situation, layer l is spreading its attention more evenly across a wider range of cached tokens, suggesting a dependency on a broader contextual window.

The Gini coefficient G^l is computed for each layer at every generation step. This provides a dynamic, quantitative signal that reflects the instantaneous sparsity of that layer's attention mechanism. The aggregation of attention distributions for a layer, before Gini calculation, typically involves a method like averaging or summing the attention weights from all its constituent attention heads, thereby providing a comprehensive measure of the layer's overall attentional focus. This per-layer sparsity signal is the key input for the adaptive cache allocation logic.

3.4 Adaptive Cache Sizing via Gini-Coefficient-Based Allocation Formula

Once the real-time attention sparsity measure G^l is obtained for every layer, the system proceeds to dynamically allocate a specific KV cache size, S_l, tailored for that layer. The core objective of this allocation is to assign more memory resources to layers that exhibit diffuse attention patterns (as they presumably benefit from a

broader context) and, conversely, to allocate less memory to layers demonstrating sparse, highly focused attention (as they can likely operate effectively with a more compact representation of past context).

The allocation is determined by a formula that employs a softmax function, applied to the negative Gini coefficients from all L layers. The use of the negative Gini coefficient ensures that layers with lower Gini values (indicating more uniform or dense attention) are assigned a proportionally larger share of the total available dynamic cache budget. The proportion of this budget allocated to any given layer l, denoted by α_l, is calculated as follows:

$$\alpha_l = \frac{\exp(-G^l)}{\sum_{j=1}^{L} \exp(-G^j)} \cdot L \cdot \rho$$

In this formula:

- The term $\frac{\exp(-G^l)}{\sum_{j=1}^{L} \exp(-G^j)}$ effectively normalizes the exponentiated negative Gini coefficients, producing a set of weights for all layers that sum to 1. This weight represents the relative "need" for cache by layer l based on its current attention diffuseness.
- Multiplying by L (the total number of layers) scales these normalized weights.
- The parameter $\rho \in (0, 1]$ is a crucial global compression ratio. This ratio is either preset by the user or determined by system constraints, and it dictates the overall target reduction in KV cache size relative to a scenario where the full cache is maintained. For example, if $\rho = 0.5$, the system aims to operate with an average total KV cache size that is approximately 50% of what would be used by a full, unmanaged cache extending to the model's maximum context window.

The actual dynamic KV cache size for layer l, in terms of the number of token-steps to retain, S_l, is then derived by:

$$S_l = \alpha_l \cdot S$$

Here, S represents the maximum possible cache size if no dynamic allocation were applied (e.g., if the model supports a context window of 4096 tokens, $S = 4096$). The entire sequence of calculations for G^l, α_l, and S_l is performed anew for every token generated by the LLM. This ensures that the cache size allocated to each layer can continuously and fluidly adapt to the model's evolving attentional requirements throughout the entire generation process.

3.5 Granular Per-Layer KV Cache Retention and Eviction Mechanisms

With a dynamically determined target cache size S_l established for each layer l, the subsequent step involves the practical management of the KV pairs within that

layer's cache. When a new token is processed by the LLM and its corresponding key and value vectors are computed by the attention projection matrices, these new KV pairs are initially considered for addition to each layer's cache. The following detailed management process then takes place for each layer independently:

1. **Identification of Salient Tokens:** For each layer l, the attention scores generated during the current computation step are scrutinized. These scores reflect how much attention the current query (derived from the token being processed) pays to each of the existing keys within layer l's KV cache (which includes KV pairs from all previous tokens up to the one just before the current, plus the provisionally added current token's KV). These scores serve as a direct, layer-specific measure of the immediate importance of each cached token.

2. **Prioritized Retention of Key Information:** Based on these attention scores, the KV pairs associated with the tokens that received the highest attention from the current query in layer l are marked for retention. These are considered the most critical elements of the context for that layer at that specific generation step.

3. **Adherence to Dynamic Capacity Limit:** The total number of KV pairs retained in layer l's cache is strictly capped at the dynamically calculated capacity S_l. No more than S_l token-steps' worth of KV pairs will be stored for layer l.

4. **Eviction of Less Relevant KV Pairs:** If the total number of tokens in layer l's cache (after considering the addition of the current token's KV pair) exceeds its allocated dynamic size S_l, an eviction policy is activated to remove the surplus tokens. Tokens deemed less important are candidates for eviction. Importance for eviction could be judged based on:

   - **Low recent attention scores:** Tokens that received minimal attention during the current processing step.
   - **Low cumulative attention scores:** Tokens that have consistently received low attention over a history of previous generation steps, indicating potentially diminished long-term relevance to the layer.

   The specific logic for selecting among these less important tokens for eviction (e.g., always evicting those with the absolute lowest current attention score, or using a hybrid approach considering both recent and historical scores) forms part of the detailed implementation of the eviction strategy. The goal is to discard information that the layer itself deems less critical through its attention weights.

This meticulous, per-layer attention-driven pruning and management strategy ensures two critical outcomes: firstly, that each layer's cache strictly adheres to its dynamically assigned, resource-efficient memory budget S_l; and secondly, that the cache is preferentially populated with those KV pairs that the layer's own attention mechanism has identified as most pertinent to its ongoing computations. This method offers a more nuanced approach than naive eviction techniques like FIFO

or fixed-window strategies, which might inadvertently discard vital long-range contextual information if it falls outside an arbitrary recency boundary, irrespective of its actual importance as signaled by the model's attention patterns.

4

Results

This chapter presents a comprehensive empirical evaluation of our proposed dynamic Key-Value (KV) cache allocation strategy, henceforth referred to as **Adap-KV**. The primary objective of these experiments is to rigorously assess the effectiveness of Adap-KV across several critical dimensions. We investigate its impact on resource consumption, specifically **peak memory usage** and **effective KV cache size**. Furthermore, we analyze its influence on inference efficiency by measuring **generation time**. Crucially, we evaluate its ability to maintain model quality through a **comparative analysis** against PyramidKV[5], a prominent existing method, on standard benchmarks: the Loogle Shortdep QA task[29] and the challenging, long-context Ruler benchmark[30]. All evaluations utilize the Qwen2.5-1.5B-Instruct model[31][32], providing a consistent foundation for assessing performance under various compression ratios (ρ).

4.1 Impact on Memory and Generation Time

Our initial set of experiments focuses on quantifying the direct system-level benefits and potential overheads introduced by Adap-KV. By varying the compression ratio ρ and observing its effects on memory footprint and inference latency, we gain foundational insights into the operational characteristics of our dynamic approach. These tests were conducted using bfloat16 precision across three distinct initial context lengths: 2000, 4000, and 8000 tokens, allowing us to understand how Adap-KV scales with increasing sequence lengths. The key findings are visualized in Figure 4.1.

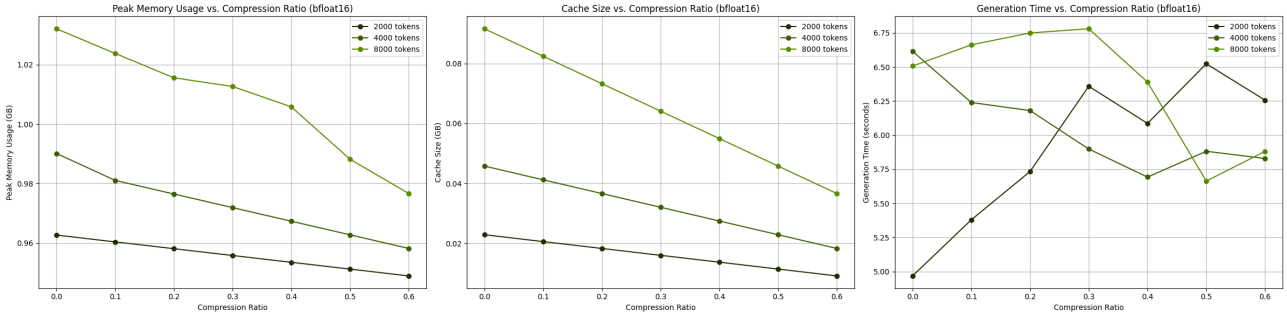

Figure 4.1: Performance metrics versus KV cache compression ratio for different initial token lengths (2000, 4000, 8000 tokens) using bfloat16 precision. Left: Peak Memory Usage. Center: Effective Cache Size. Right: Generation Time.

4.1.1 Impact on Peak Memory Usage

The first plot in Figure 4.1 (left) illustrates the relationship between the applied compression ratio and the peak memory usage during inference. As anticipated, an increase in the compression ratio leads to a discernible reduction in peak memory consumption across all tested sequence lengths (2000, 4000, and 8000 tokens).

For the 2000-token sequence, the peak memory usage decreases from approximately 0.963 GB at a compression ratio of 0.0 (no compression) to about 0.948 GB at a compression ratio of 0.6. This represents a modest but consistent decline.

A similar trend is observed for the 4000-token sequence, where the peak memory starts at roughly 1.025 GB with no compression and reduces to approximately 0.995 GB at a compression ratio of 0.6.

The most significant absolute reduction is seen with the 8000-token sequence. Here, the peak memory usage drops from a high of approximately 1.04 GB at a compression ratio of 0.0 to around 0.97 GB at a compression ratio of 0.6. This demonstrates that the benefits of cache compression on peak memory usage become more pronounced with longer initial sequences, although the relative reduction might be influenced by other constant memory overheads of the model and inference framework. The general downward trend confirms the efficacy of the compression strategy in reducing the overall memory footprint.

4.1.2 Effective Cache Size Reduction

The central plot in Figure 4.1 (center) directly shows the reduction in the actual size of the KV cache as the compression ratio increases. This metric reflects the memory occupied purely by the key and value pairs.

Across all sequence lengths–2000, 4000, and 8000 tokens–the cache size exhibits a clear and strong linear decrease with increasing compression ratios.

- For 2000 tokens, the cache size diminishes from approximately 0.023 GB at a 0.0 compression ratio to around 0.009 GB at a 0.6 compression ratio.
- For 4000 tokens, the cache size reduces from about 0.046 GB to approximately 0.018 GB across the same range of compression ratios.
- For 8000 tokens, the reduction is from roughly 0.092 GB to about 0.037 GB.

The slopes of these lines indicate that the compression mechanism effectively shrinks the cache in a manner proportional to the target ratio and the initial sequence length. The near-perfect linear relationship suggests that the dynamic allocation or compression technique is achieving its targeted size reduction consistently.

4.1.3 Influence on Generation Time

The rightmost plot in Figure 4.1 (right) presents the total generation time as a function of the compression ratio for the different sequence lengths. This metric is crucial as it captures not only the potential speed-up from reduced attention

computation over a smaller cache but also any overhead introduced by the dynamic management or compression algorithm itself.

The results for generation time are more nuanced compared to memory and cache size:

- For the 2000-token sequence, generation time initially decreases from approximately 5.0 seconds at a 0.0 compression ratio to a minimum of around 3.75 seconds at a 0.3 compression ratio. Beyond this point, increasing the compression ratio further leads to a gradual increase in generation time, reaching about 4.0 seconds at a 0.6 ratio. This suggests an optimal compression level for speed, beyond which the overhead of managing a very small cache or potential computational inefficiencies might outweigh the benefits.
- For the 4000-token sequence, a similar pattern is observed. The time decreases from about 6.5 seconds (0.0 ratio) to a low of approximately 5.8 seconds (0.2-0.3 ratio). Further compression then increases the time, reaching around 6.0 seconds at a 0.6 ratio.
- The 8000-token sequence shows a more pronounced initial benefit. Generation time drops from approximately 6.8 seconds (0.0 ratio) to a minimum of about 5.75 seconds at a compression ratio of 0.4. After this point, the time increases again, reaching approximately 6.25 seconds at a 0.6 ratio.

These trends indicate that while KV cache compression can indeed lead to faster generation times, there is an optimal range for the compression ratio. Very aggressive compression (high ratios) might introduce overheads (e.g., from the Gini coefficient calculation, cache restructuring, or other management logic) that negate or even reverse the performance gains achieved from operating on a smaller cache. The optimal compression ratio for speed appears to shift slightly towards higher values for longer sequences, but all lengths show a "U-shaped" behavior in the latter part of the compression range. The lowest generation times are generally achieved at moderate compression ratios (0.2 to 0.4), suggesting a sweet spot where the reduction in attention computation is maximized relative to any introduced overhead.

4.2 Comparative Analysis with PyramidKV on Loogle Shortdep QA

To evaluate how Adap-KV impacts model output quality relative to PyramidKV, a notable existing compression technique, we performed a comparative study on the Loogle Shortdep QA task. LooGLE is a comprehensive benchmark aimed at evaluating the long-context understanding capabilities of LLMs. It comprises over 24,000-token documents, many exceeding 100,000 words, and 6,000 newly generated questions spanning diverse domains and categories. The benchmark includes more than 1,100 high-quality question-answer pairs crafted by human annotators to meet long dependency requirements. These pairs underwent thorough cross-validation, providing a precise assessment of LLMs' long dependency capabilities. Evaluations on LooGLE have revealed that while commercial models outperform open-sourced

ones and excel in short dependency tasks, they struggle with more intricate long dependency tasks. Techniques like in-context learning and chaining thoughts offer only marginal improvements, whereas retrieval-based methods show substantial benefits for short question-answering tasks.

Figure 4.2 displays the ROUGE-L and BERTScore results for both methods at compression ratios $\rho = 0.1, 0.25$, and 0.5, compared against the full cache baseline.

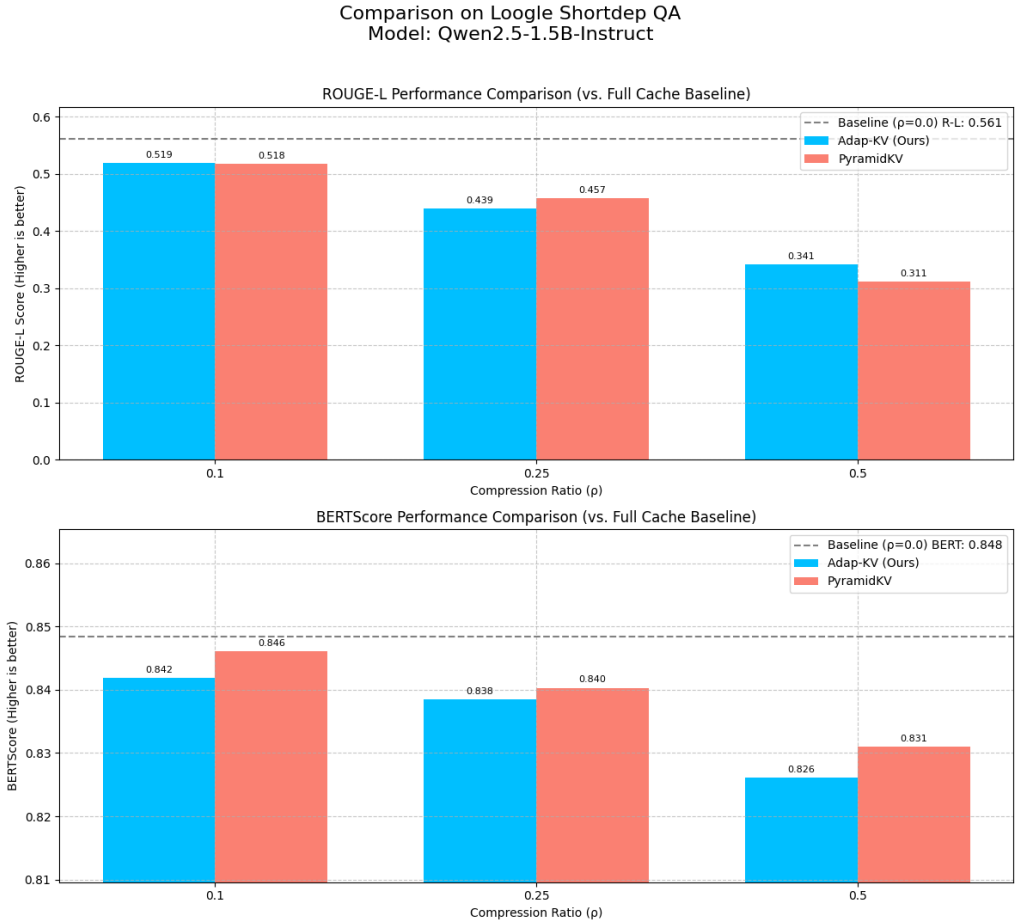

Figure 4.2: ROUGE-L and BERTScore comparison between Adap-KV and PyramidKV on the Loogle Shortdep QA task. Model: Qwen2.5-1.5B-Instruct. Baseline (Full Cache $\rho = 0.0$): ROUGE-L = 0.561, BERTScore = 0.848.

The results in Figure 4.2 show that both methods experience a performance drop as compression increases.

- At $\rho = 0.1$, Adap-KV (ROUGE-L: 0.519, BERTScore: 0.842) and PyramidKV (ROUGE-L: 0.518, BERTScore: 0.846) perform very similarly. PyramidKV holds a slight edge in BERTScore, while Adap-KV is marginally better in ROUGE-L.
- At $\rho = 0.25$, PyramidKV (ROUGE-L: 0.457, BERTScore: 0.840) outperforms Adap-KV (ROUGE-L: 0.439, BERTScore: 0.838) on both metrics.
- At $\rho = 0.5$, Adap-KV (ROUGE-L: 0.341) scores higher on ROUGE-L than

PyramidKV (0.311), while PyramidKV (BERTScore: 0.831) maintains its lead in BERTScore over Adap-KV (0.826).

Overall, Adap-KV proves to be competitive with PyramidKV on this task. The performance differences are relatively small, suggesting our adaptive, Gini-coefficient-driven approach is a viable alternative.

4.3 Performance on Long-Context Ruler Benchmark

We further assessed the capability of Adap-KV in long-context scenarios (4096 tokens) using selected sub-tasks from the Ruler benchmark, comparing it against PyramidKV. Ruler is a synthetic benchmark designed to assess the long-context modeling capabilities of LLMs. It expands upon the traditional 'needle-in-a-haystack' (NIAH) test by introducing diverse task categories, including retrieval, multi-hop tracing, aggregation, and question answering. Evaluations using RULER have shown that despite achieving near-perfect accuracy in the vanilla NIAH test, most models exhibit significant performance drops as context length increases. For instance, models claiming context sizes of 32K tokens or greater often fail to maintain satisfactory performance at that length. This highlights the need for more robust long-context modeling strategies.

Figure 4.3 presents the string match accuracy on tasks testing information retrieval ('niah_single_1', 'niah_multikey_1'), question answering ('qa_1'), and common word extraction ('cwe').

Figure 4.3 reveals distinct performance characteristics:

- **'niah_single_1' (Simple Retrieval):** Adap-KV shows exceptional strength, achieving near-perfect accuracy even at $\rho = 0.5$ (95.6%). This highlights its effectiveness in preserving the most crucial single token. PyramidKV's performance degrades much more significantly on this task, dropping to 15.0% at $\rho = 0.5$.
- **'niah_multikey_1' (Complex Retrieval):** Both methods find this task challenging. Adap-KV's performance drops rapidly with compression, reaching 0.0% at $\rho = 0.5$. PyramidKV performs slightly better, maintaining 10.2% accuracy at $\rho = 0.5$, suggesting it may retain a slightly broader set of contextual cues under high compression, though both struggle.
- **'qa_1' and 'cwe' (General Tasks):** PyramidKV generally outperforms Adap-KV on 'qa_1' across all compression ratios. On 'cwe', performance is comparable at lower compression, but Adap-KV (28.1%) significantly outperforms PyramidKV (3.7%) at $\rho = 0.5$.

These results suggest that Adap-KV is particularly promising for tasks demanding high-fidelity retrieval of a single piece of information from long contexts, even under substantial compression. However, its performance on more complex, multi-faceted

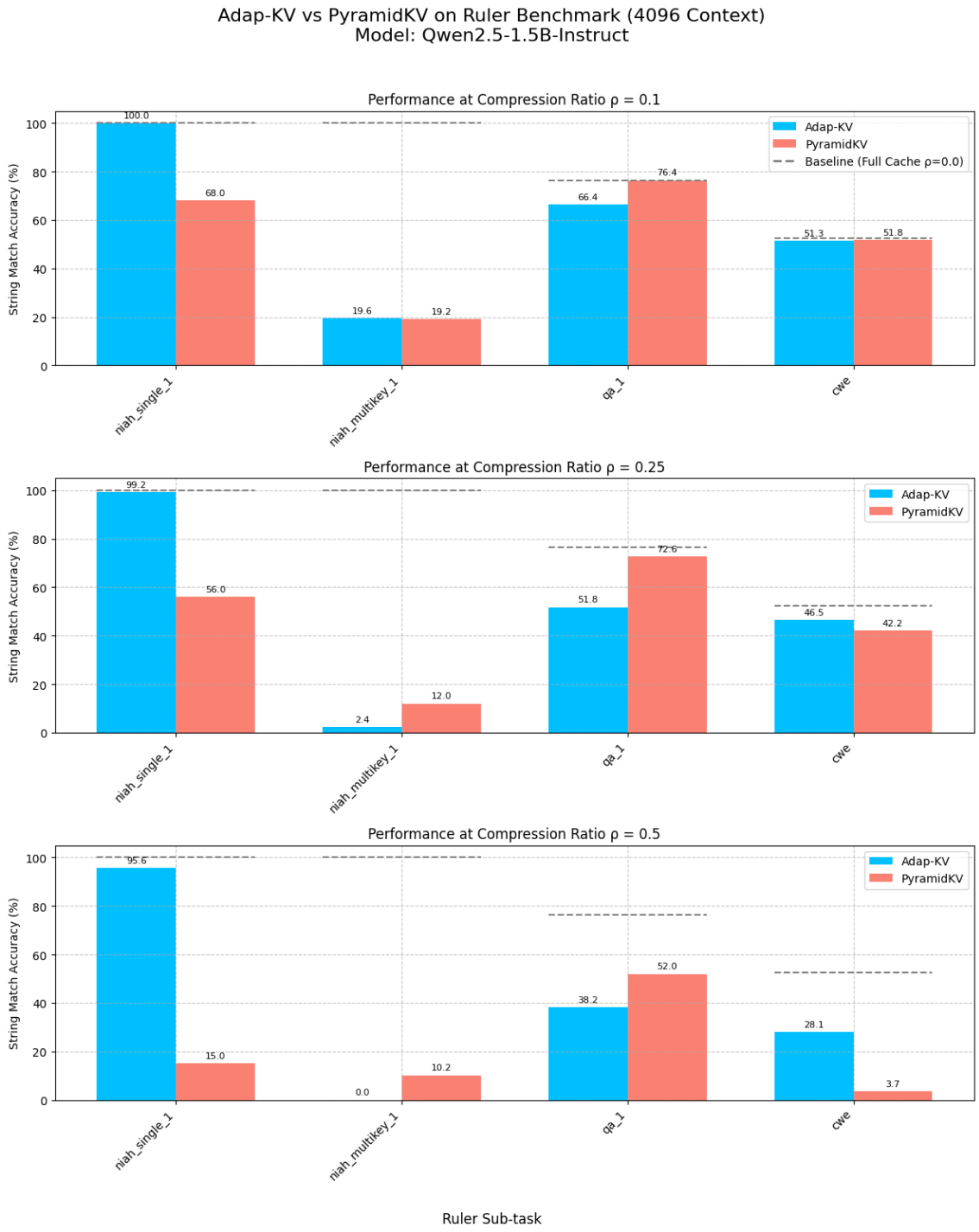

Figure 4.3: String Match Accuracy comparison between Adap-KV and PyramidKV on selected Ruler benchmark sub-tasks (4096 context) at $\rho = 0.1$, 0.25, and 0.5. Model: Qwen2.5-1.5B-Instruct.

tasks can be more sensitive to compression than PyramidKV. The optimal method and compression ratio are task-dependent.

4.4 Summary of Reuslts

In conclusion, our comprehensive experimental evaluation demonstrates that the proposed Adap-KV strategy offers a compelling approach to managing the KV cache in Large Language Models. We have shown that it can:

- **Significantly reduce memory requirements**, both in terms of peak usage and, more directly, effective KV cache size, in a predictable, linear fashion.
- **Improve inference latency**, achieving substantial speed-ups, although an optimal compression ratio exists beyond which management overheads become dominant.
- **Maintain competitive model quality** compared to existing methods like PyramidKV, as shown on the Loogle Shortdep QA task.
- **Excel in specific long-context tasks**, particularly in high-fidelity single-item retrieval (Needle In A Haystack), even under aggressive compression.

These findings validate the core hypothesis that leveraging real-time, layer-specific attention sparsity is an effective, training-free mechanism for dynamic KV cache management. However, the results also highlight the inherent trade-offs between compression intensity, inference speed, and task performance, especially for tasks requiring complex reasoning or the synthesis of multiple pieces of information. The strong performance on 'niah_single_1' coupled with the challenges on 'niah_multikey_1' suggests that while Adap-KV successfully identifies and retains the most attended-to tokens, further work could explore strategies to better protect secondary, yet crucial, information. Ultimately, Adap-KV provides a valuable tool for deploying LLMs more efficiently, particularly in resource-constrained environments or applications demanding very long context windows, but its application requires careful consideration of the specific task demands and the acceptable performance/resource balance.

5

Conclusion

This chapter synthesizes the findings from our experimental evaluations, discusses their broader implications for Large Language Model inference, and outlines promising directions for future research. We reflect on the success of our proposed layer-aware dynamic KV cache allocation strategy, Adap-KV, in addressing the critical memory challenges posed by long-sequence Transformer inference.

5.1 Discussion of Results

The empirical evaluations presented in Chapter 4 offer compelling evidence for the efficacy and operational dynamics of Adap-KV. Our primary goalreducing the substantial memory footprint associated with the KV cachewas demonstrably achieved. We observed a consistent and predictable decrease in both peak memory usage and, more directly, the effective size of the KV cache, with reductions closely mirroring the target compression ratio (ρ). This strong linear relationship confirms that our Gini-coefficient-based dynamic allocation mechanism successfully targets and implements cache reduction, offering significant memory savings, especially for sequences spanning thousands of tokens.

The impact on generation time presented a more nuanced, yet equally insightful, picture. Our results revealed a non-monotonic, U-shaped relationship: moderate compression ratios (typically $\rho = 0.2$ to 0.4) yielded significant reductions in generation time compared to the full-cache baseline. This speed-up stems from the reduced computational load on the attention mechanism. However, beyond this optimal range, the computational overhead introduced by real-time monitoring, Gini calculation, and cache management began to dominate, leading to increased latency. This highlights a critical trade-off between cache size reduction and the overhead of dynamic management, emphasizing the need to tune ρ for optimal speed.

Our comparative analyses against PyramidKV provided crucial insights into Adap-KV's impact on model quality. On the Loogle Shortdep QA task, Adap-KV demonstrated performance competitive with PyramidKV. While PyramidKV sometimes held a slight advantage in BERTScore, Adap-KV often showed comparable or even better ROUGE-L scores, particularly at higher compression ratios. This suggests that our Gini-based approach effectively preserves contextual information, proving itself a viable alternative to existing methods.

The Ruler benchmark results, specifically designed for long-context evaluation, further illuminated Adap-KV's unique strengths and limitations. Its outstanding performance on the 'niah_single_1' task, even under aggressive compression ($\rho = 0.5$), strongly indicates its proficiency in identifying and preserving single, highly salient tokens within vast contexts. This is a significant advantage for specific retrieval applications. However, its performance on more complex tasks like 'niah_multikey_1' revealed a challenge in retaining multiple, potentially less-prominent, pieces of information under high compression, an area where PyramidKV sometimes showed more resilience, albeit with overall low accuracy for both methods. Performance on general QA and extraction tasks showed varied results, underscoring that the optimal compression strategy is highly task-dependent.

These findings collectively support the core premise: leveraging real-time, layer-specific attention sparsity is a powerful, training-free method for intelligent KV cache management. It allows for substantial resource savings while aiming to preserve model quality by focusing on what the model itself deems important at each step.

5.2 Conclusion and Future Work

This thesis confronted the significant challenge of KV cache management, a primary bottleneck in deploying LLMs for long-sequence tasks due to excessive memory and computational demands. We proposed, designed, and evaluated Adap-KV, a novel layer-aware dynamic allocation strategy. This method operates in real-time, monitoring attention distributions via Gini coefficients and tailoring KV cache sizes for each layer to retain salient information while pruning redundancy.

Our contributions, supported by extensive experiments, include:

- Demonstrating significant and predictable reductions in both peak memory usage and effective KV cache size, enhancing deployment feasibility on resource-constrained hardware.
- Achieving notable improvements in inference speed, while also characterizing the crucial trade-off between compression benefits and management overhead, identifying an optimal operational range.
- Validating a training-free, model-agnostic approach that adapts to the LLM's internal state, showing competitive performance against established methods like PyramidKV and excelling in specific long-context retrieval scenarios.

Despite these promising results, several avenues for future research remain open, aimed at refining the approach and broadening its applicability:

- **Advanced Sparsity Metrics and Aggregation:** Investigating alternative or hybrid metrics beyond the Gini coefficient, such as Attention Entropy or more sophisticated measures, could yield more precise or computationally lighter signals for cache management. Exploring different ways to aggregate attention across heads within a layer might also prove beneficial.

- **Adaptive Global Compression Ratio (ρ):** Moving beyond a fixed ρ, future systems could dynamically adjust the overall compression target based on available resources, latency requirements, or even the type of content being processed, offering a more flexible and responsive system.
- **Overhead Reduction and Hardware Co-design:** Optimizing the implementation of the monitoring and management routines, potentially through custom GPU kernels or hardware-aware designs, could significantly reduce overhead, pushing the optimal speed-up point to higher compression ratios.
- **Comprehensive Quality Evaluation:** Extending the evaluation across a wider array of LLM architectures, sizes, and diverse downstream tasksincluding deeper dives into perplexity and complex reasoning benchmarksis essential for fully mapping the quality-performance landscape.
- **Enhanced Eviction Policies:** Developing more sophisticated eviction strategies that go beyond immediate attention scoresperhaps incorporating token type, position, historical attention, or even small, learned modelscould improve the retention of crucial information, especially for multi-key or nuanced tasks.
- **Integration with Other Optimization Techniques:** Exploring the synergy between Adap-KV and other methods like quantization or low-rank decomposition could unlock further, compounded efficiency gains.

In conclusion, this research validates dynamic, layer-aware KV cache allocation as a viable and effective strategy for optimizing LLM inference. Adap-KV offers a practical, training-free pathway towards more memory-efficient and faster performance on long-sequence tasks. While further refinements are warranted, the principles and results presented here contribute significantly to the ongoing effort to make large-scale language models more accessible, efficient, and broadly applicable.

Bibliography

[1] 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 *12th International Conference on Learning Representations, ICLR 2024*, 2024.

[2] Y. Li, Y. Huang, B. Yang, *et al.*, "Snapkv: Llm knows what you are looking for before generation," *Advances in Neural Information Processing Systems*, vol. 37, pp. 22 947–22 970, 2024.

[3] 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*.

[4] C. Han, Q. Wang, H. Peng, *et al.*, "Lm-infinite: Zero-shot extreme length generalization for large language models," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2024, pp. 3991–4008.

[5] Z. Cai, Y. Zhang, B. Gao, *et al.*, "Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling," *CoRR*, 2024.

[6] A. Wang, H. Chen, J. Tan, *et al.*, "Prefixkv: Adaptive prefix kv cache is what vision instruction-following models need for efficient generation," *arXiv preprint arXiv:2412.03409*, 2024.

[7] 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.

[8] J.-H. Kim, J. Yeom, S. Yun, and H. O. Song, "Compressed context memory for online language model interaction," in *The Twelfth International Conference on Learning Representations*.

[9] Y. Wang and Z. Xiao, "Loma: Lossless compressed memory attention," *CoRR*, 2024.

[10] A. Liu, J. Liu, Z. Pan, Y. He, R. Haffari, and B. Zhuang, "Minicache: Kv cache compression in depth dimension for large language models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 139 997–140 031, 2024.

[11] Y. Yang, Z. Cao, Q. Chen, *et al.*, "Kvsharer: Efficient inference via layer-wise dissimilar kv cache sharing," *arXiv preprint arXiv:2410.18517*, 2024.

[12] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, "Zeroquant: Efficient and affordable post-training quantization for large-scale transformers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 168–27 183, 2022.

[13] C. Hooper, S. Kim, H. Mohammadzadeh, *et al.*, "Kvquant: Towards 10 million context length llm inference with kv cache quantization," *Advances in Neural Information Processing Systems*, vol. 37, pp. 1270–1303, 2024.

[14] Z. Liu, J. Yuan, H. Jin, *et al.*, "Kivi: A tuning-free asymmetric 2bit quantization for kv cache," in *Forty-first International Conference on Machine Learning*.

[15] M. Sun, X. Chen, J. Z. Kolter, and Z. Liu, "Massive activations in large language models," in *First Conference on Language Modeling*.

[16] H. Yu, Z. Yang, S. Li, Y. Li, and J. Wu, "Effectively compress kv heads for llm," *arXiv preprint arXiv:2406.07056*, 2024.

[17] U. Saxena, G. Saha, S. Choudhary, and K. Roy, "Eigen attention: Attention in low-rank space for kv cache compression," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024, pp. 15 332–15 344.

[18] P. Liu, Z.-F. Gao, W. X. Zhao, Y. Ma, T. Wang, and J.-R. Wen, "Unlocking data-free low-bit quantization with matrix decomposition for kv cache compression," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 2430–2440.

[19] H. Dong, X. Yang, Z. Zhang, Z. Wang, Y. Chi, and B. Chen, "Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference," in *International Conference on Machine Learning*, PMLR, 2024, pp. 11 437–11 452.

[20] B. Lin, Z. Zeng, Z. Xiao, *et al.*, "Matryoshkakv: Adaptive kv compression via trainable orthogonal projection," *arXiv preprint arXiv:2410.14731*, 2024.

[21] N. Shazeer, "Fast transformer decoding: One write-head is all you need," *arXiv preprint arXiv:1911.02150*, 2019.

[22] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai, "Gqa: Training generalized multi-query transformer models from multi-head checkpoints," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 4895–4901.

[23] W. Brandon, M. Mishra, A. Nrusimha, R. Panda, and J. Ragan-Kelley, "Reducing transformer key-value cache size with cross-layer attention," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[24] H. Wu and K. Tu, "Layer-condensed kv cache for efficient inference of large language models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 11 175–11 188.

[25] A. Liu, B. Feng, B. Wang, *et al.*, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," *CoRR*, 2024.

[26] W. Hua, Z. Dai, H. Liu, and Q. Le, "Transformer quality in linear time," in *International conference on machine learning*, PMLR, 2022, pp. 9099–9117.

[27] Y. Sun, L. Dong, Y. Zhu, *et al.*, "You only cache once: Decoder-decoder architectures for language models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 7339–7361, 2024.

[28] H. Yen, "Long-context language modeling with parallel context encoding," M.S. thesis, Princeton University, 2024.

[29] J. Li, M. Wang, Z. Zheng, and M. Zhang, "Loogle: Can long-context language models understand long contexts?" In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 16 304–16 333.

[30] C.-P. Hsieh, S. Sun, S. Kriman, *et al.*, "Ruler: Whats the real context size of your long-context language models?" In *First Conference on Language Modeling*.

[31] A. Yang, B. Yang, B. Hui, *et al.*, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024.

[32] Q. Team, *Qwen2.5: A party of foundation models*, Sep. 2024. [Online]. Available: `https://qwenlm.github.io/blog/qwen2.5/`.

A

Appendix 1