LuKA: Line and Cluster Key-Value Approximations

Anthony Fei, Rolando Rodríguez, Krishna Patel, Gerardo Montemayor
Cornell University

Project GitHub: https://github.com/ayf7/LuKA

Abstract

As large language models (LLMs) are increasingly deployed for complex tasks that require extensive context, the quadratic scaling of the attention mechanism presents a significant computational bottleneck. The key-value (KV) cache is a central inference-time mechanism for addressing this problem, storing past token representations to avoid redundant inference-time computation, but grows linearly with sequence length, often exceeding available GPU memory. Methods for KV-cache optimization have recently been shown to perform well while decreasing the attention computation, but typically drop entire sequences that are labeled unimportant, leading to permanent loss of information.

We introduce LuKA, a structured KV cache design that compresses "pages" of adjacent tokens into summary representations while allowing selective refinement to recover original tokens KVs when needed. LuKA achieves promising results, improving over H_2O on QA, potentially offering better performing approximations. LuKA still remains a proof-of-concept, as it faces limitations in memory and compute as well as significant throughput overhead compared to vanilla attention, which we leave for optimization in further work.

1 Introduction

The deployment of Large Language Models (LLMs) in real-world applications is increasingly constrained by the computational and memory cost of processing long contexts. While the key-value (KV) cache allows models to avoid recomputing representations for past tokens, it introduces a linearly scaling growth in memory usage and attention complexity. As context windows extend to hundreds of thousands of tokens, two bottlenecks emerge: memory usage grows linearly with sequence length, and attention computation becomes increasingly expensive.

Current approaches to this problem, such as Heavy-Hitter Oracle (H_2O) [7] and StreamingLLM [5], rely on *eviction* policies. These methods define heuristics to identify "unimportant" tokens and permanently remove them from memory. While effective for maintaining fixed memory budgets, eviction is an irreversible operation; if a token is evicted, it cannot be retrieved, even if it becomes critical in the future. While these token keys and values could in theory be recomputed, in order to know that this would need to happen they need to first be attended to, a Catch-22 for eviction policy KV caches. This limitation is particularly damaging when dealing with long contexts (e.g., extended chat-based conversations, textbooks) that are full of information that may be mutually independent: What is at one time irrelevant to a chatter's first query about, say, Bellman-Ford algorithm and dynamic programming, may be very relevant to their next query about universal Turing machines. While eviction policies can be understood as targeting memory improvements, they are also asking the fundamental question of which tokens are important and which aren't– in this sense, they are also just as much about improving attention mechanism efficiency.

To address this, we introduce **LuKA**, a structured KV cache design that takes a different approach towards improving attention efficiency over long contexts. Instead of discarding data, LuKA compresses segments of contiguous tokens ("pages") of context into compact summary representations. Critically, LuKA maintains a mechanism for *selective refinement*: When the model's attention mechanism indicates that a compressed summary is relevant to the current query, LuKA dynamically unrolls the summary into its original tokens. It is important to note that LuKA is consequently not primarily optimized for minimal KV cache memory footprint.

Algebraic topology studies topological spaces using tools from abstract algebra, such as groups and rings, to capture their fundamental structural properties. It focuses on concepts like homotopy, homology, and cohomology, which help distinguish spaces by their shapes and connectivity rather than geometric details. This field bridges pure mathematics and geometry, providing insights relevant to modern physics and data analysis.

World War II was a global conflict from 1939 to 1945 that reshaped political, social, and economic landscapes worldwide. It involved the major Allied and Axis powers and introduced unprecedented warfare technologies, from radar to nuclear weapons. The aftermath led to the rise of the United States and the Soviet Union as superpowers and set the stage for the Cold War.

Building on these foundations, modern research in algebraic topology explores connections with category theory, quantum field theory, and even data science. Concepts such as persistent homology and topological data analysis extend classical ideas to study the structure of high-dimensional datasets, revealing patterns that traditional statistical tools might miss. This synthesis of abstract theory and real-world application illustrates how topology continues to evolve as both a theoretical and practical discipline.

What defines something as truly innovative?

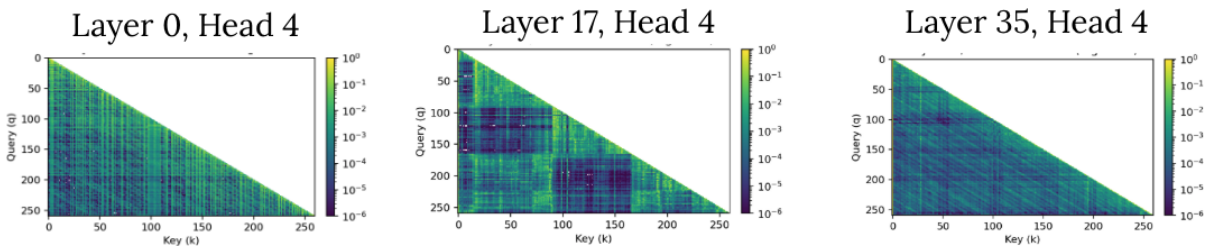

Figure 1: **Interwoven Prompt and Resulting Attention Pattern.** In the prompt above, there are three completely separate topics: algebraic topology, World War II, and a question regarding innovation. The attention map highlights how the model connects distant segments of the same topic.

Instead, it explicitly trades-off the aggressive memory savings of hard eviction for information fidelity, prioritizing a more efficient attention mechanism without the risk of permanent information loss.

We aim to demonstrate that LuKA reduces the expected computation cost of long-context inference from raw attention while maintaining higher accuracy than eviction-based baselines.

2 Related Work & Motivation

Recent works in KV cache optimization introduce strict heuristics for dropping tokens deemed unimportant for future decoding steps. H_2O retains a subset of past tokens based on attention-based importance scores, pruning the KV cache to keep high-attention tokens. SnapKV [1] compresses KV caches by selecting clustered important KV positions per attention head. StreamingLLM uses a fixed sliding window, dropping tokens as they fall out of range, and relies on attention "sink" tokens to stabilize long-context computation. While these approaches demonstrate that reducing the KV cache can preserve performance, several share a key limitation: once information is dropped, it cannot be recovered. Other approaches introduce a hierarchy in addition to token eviction policies, such as Quest [4], which reduces KV-cache cost by selecting a limited subset of tokens for attention based on the current query. Native Sparse Attention [6] combines hierarchical compression with selective token retention and sliding windows, learning sparse attention patterns end-to-end rather than relying on fixed eviction rules. In contrast, LuKA is motivated by observed layer-wise attention patterns and applies different attention methods across layers, rather than learning a single unified sparse method.

Our work is motivated by observations of structured attention patterns in long-context prompts. In early experiments using interwoven prompts containing multiple topics (Figure 1), we visualized attention maps across layers and observed two distinct behaviors. As illustrated, the topic of algebraic topology appears twice within the same prompt, and attention in Layer 17, Head 4 strongly connects these two distant text segments. We refer to this behavior as *clustered attention*, where semantically related regions of the context attend to each other despite being far apart. In contrast, earlier and later layers exhibit *lined attention*, where high attention activations are tied to specific token positions rather than semantic groupings. These observations motivate a design that treats these patterns differently.

3 Methodology

LuKA is a hybrid approach that provides different sparse attention methods to different attention layers in a transformer. As mentioned previously,

we denote these as lined attention and clustered attention.

3.1 Lined Attention (H_2O)

Lined attention follows the Heavy-Hitter Oracle (H_2O) approach for sparse attention, which preserves individual tokens that receive consistently high attention over time. Rather than compressing tokens, lined attention maintains a fixed-size sliding window consisting of heavy hitter tokens and a sliding window of recent tokens, both without compression.

At each decoding step, LuKA selects a fixed fraction of past tokens with the highest accumulated attention scores as heavy hitters and combines them with a sliding window of recent tokens to form the attention cover, removing any duplicates. Attention is then computed only over this reduced set.

Attention scores are accumulated without decay across decoding and prefill, following the H_2O mechanism, ensuring that heavy-hitter selection reflects true attention patterns. By attending to only a fixed fraction of the sequence, lined attention reduces the effective attention span while preserving globally important tokens.

3.2 Clustered Attention

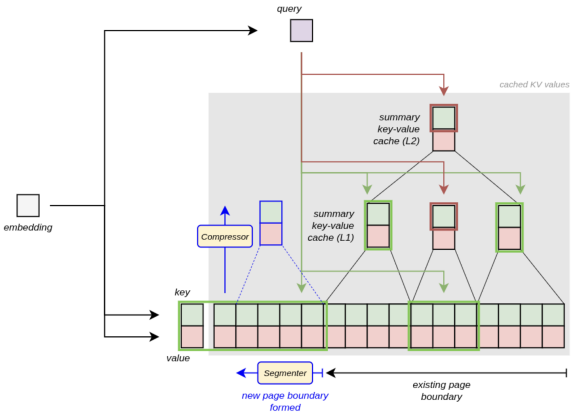

Figure 2: LuKA clustered attention mechanism.

Clustered attention reduces attention cost by compressing older portions of the context into summary pages while preserving equivalence to full attention. The key idea is to organize the KV cache into a hierarchy and to perform attention in a top-down manner, refining compressed representations only when necessary, illustrated in Figure 2. The pseudocode for the algorithm can be found below.

Hierarchical KV Organization. Let $\mathcal{K}^{\text{raw}} = \{(k_i, v_i)\}_{i=1}^{T}$ denote the standard KV cache. Older, contiguous segments of tokens are optionally compressed into *summary nodes*, each of which represents a page of underlying tokens. These summaries may themselves be recursively compressed, forming a tree-structured hierarchy $\mathcal{T}$ whose leaves correspond to raw tokens.

Each node $n \in \mathcal{T}$ is associated with a key–value pair (k_n, v_n), where summary nodes are produced by a learned compressor. For any time step, we define an *attention cover* $\mathcal{C} \subset \mathcal{T}$ as a set of nodes such that (i) every raw token is represented by exactly one node in $\mathcal{C}$ (either itself or an ancestor), and (ii) no node in $\mathcal{C}$ has a parent also in $\mathcal{C}$. Attention is computed only over this cover.

(Top-Down) Clustered Attention Computation. Given a query q, attention is first computed over the cover:

$$\ell_c = q^\top k_c, \qquad \alpha_c = \frac{\exp(\ell_c)}{\sum_{c' \in \mathcal{C}} \exp(\ell_{c'})}, \quad (1)$$

where α_c denotes the total attention mass assigned to node c.

If c corresponds to a raw token, its contribution $\alpha_c v_c$ is added directly to the output. If c is a summary node, a refinement rule R determines whether c should be expanded. If refinement is not triggered, the summary value v_c is used as-is. Otherwise, attention is recursively redistributed over the children of c.

Refinement Strategy. When a node $c \in \mathcal{C}$ corresponds to a summary, a refinement rule R determines whether it should be expanded into its children. We consider the following refinement strategies:

- *Fixed-Threshold:* refine c if its attention mass satisfies $\alpha_c > \epsilon$.
- *Top-K:* refine the K summary nodes with the largest attention masses.
- *Top-Fraction:* refine a fixed fraction ρ of summary nodes with the highest attention masses.

Mass-Preserving Refinement. Let $\{c_1, \ldots, c_K\}$ denote the children of a summary node c selected for refinement. Child logits are computed as $\ell_i = q^\top k_{c_i}$ and normalized locally:

$$\beta_i = \frac{\exp(\ell_i)}{\sum_{j=1}^{K} \exp(\ell_j)}. \quad (2)$$

Crucially, these local proportions do not define new attention mass. Instead, each child receives a share of its parent's mass:

$$\alpha_{c_i} = \alpha_c \cdot \beta_i. \tag{3}$$

Thus, refinement redistributes existing attention mass rather than introducing a new normalization. This invariant ensures that clustered attention is equivalent to full attention over all raw keys, up to the approximation quality of the summary representations.

Output Construction. The final attention output is the sum of contributions from all resolved nodes:

$$o = \sum_{n \in \mathcal{R}} \alpha_n v_n, \tag{4}$$

where $\mathcal{R}$ denotes the set of nodes reached after recursively applying refinement decisions.

Dynamic Segmentation and Compression. Independently of attention computation, a segmenter identifies contiguous spans of raw tokens that are eligible for compression during decoding. When triggered, a compressor aggregates each segment into a new summary node, which is inserted into the hierarchy and replaces its constituent tokens in future attention covers. Recent tokens are typically excluded from compression.

While our method proposes multiple levels of hierarchy, our experiments only explore one level of summary keys and values. Moreover, new pages are always created starting from the oldest token that is not represented by a summary.

Summary Compression Functions. When a contiguous segment of tokens $\{(k_i, v_i)\}_{i=1}^{L}$ is selected for compression, a compressor produces a single summary key–value pair $(k_{\text{sum}}, v_{\text{sum}})$ that replaces the segment in future attention covers. We evaluate the following compressors:

- *Mean:* Uniformly averages keys and values:

$$k_{\text{sum}} = \frac{1}{L} \sum_{i=1}^{L} k_i, \qquad v_{\text{sum}} = \frac{1}{L} \sum_{i=1}^{L} v_i.$$

- *Attention-Weighted:* Weights tokens by accumulated importance scores w_i (e.g., past attention mass), normalized with a softmax to provide an affine combination of the keys:

$$\pi_i = \frac{\exp(w_i/\tau)}{\sum_j \exp(w_j/\tau)},$$
$$k_{\text{sum}} = \sum_i \pi_i k_i,$$
$$v_{\text{sum}} = \sum_i \pi_i v_i.$$

Algorithm 1 Clustered Attention with Hierarchical Summary Refinement

Require: Query q; hierarchical KV tree $\mathcal{T}$; refinement rule R; segmenter S; compressor C.

function CLUSTEREDATTENTION($q, \mathcal{T}$)
  Let $\mathcal{C}$ be the *cover* of $\mathcal{T}$ ▷ each raw token represented once
  Compute logits $\ell_c \leftarrow q \cdot k_c$ for all $c \in \mathcal{C}$
  Compute attention masses $\alpha_c \leftarrow \mathrm{softmax}(\{\ell_c\}_{c \in \mathcal{C}})$
  Initialize output $o \leftarrow 0$
  for all $c \in \mathcal{C}$ **do**
    if c is a summary node **and** $R(q, c, \alpha_c) =$ `refine` **then**
      Retrieve children $\{c_i\}$ of c
      Run attention on $\{c_i\}$ **such that** total softmax mass $= \alpha_c$
      Accumulate child contributions into o
    else
      $o \leftarrow o + \alpha_c \cdot v_c$
    end if
  end for
  if S signals a new segment in the raw keys **then**
    Compress segment using C and update $\mathcal{T}$
  end if
  return o
end function

The importance score w_i is defined as the accumulated attention mass received by token i:

$$w_i = \sum_{t \in \mathcal{Q}} \alpha_{t,i},$$

This biases summaries toward consistently high importance tokens, which aligns thematically to methods like H_2O.

- *Random:* Selects a single token uniformly at random from the segment and uses its key and value as the summary. This serves as a diagnostic baseline to test whether structured aggregation is necessary.

All compressors introduce approximation only through representation; any loss of fidelity can be corrected by refinement, which redistributes the summary's attention mass back to its constituent tokens when needed.

3.3 Attention Configuration Across Layers

Lined and clustered attention are *per-layer* algorithms, so within a transformer model we can decide which type of attention to run per layer. In our experiments, we consider three modes: (1) *lined-only*, where every single attention block is given H_2O attention, (2) *clustered-only*, where every single attention block is given clustered attention, and (3) *mixed*, where the beginning and end layers are given lined, and the intermediate layers are given clustered.

4 Experimental Setup

4.1 WikiSalad Dataset

Existing long-context benchmarks evaluate models on tasks where the entire context remains relevant uniformly all through generation. While sufficient for many long-context tasks, we felt that these benchmarks lacked the multifacetedness in context that would test LuKA's selective retrieval mechanism. For that reason, another contribution of this paper is the **WikiSalad** dataset, constructed from the Stanford Question Answering Dataset (SQuAD v2.0) [3] that itself contains 100K+ human-annotated question-answer pairs on 500+ Wikipedia articles. For each evaluation example, WikiSalad samples N random articles from SQuaD, extracts context paragraphs and their associated questions, and interleaves them according to predefined patterns. This creates prompts with independent, alternating topics that are orthogonal in content, allowing for us to test whether LuKA can maintain QA accuracy across interlaced topics with selective retrieval.

We generated four difficulty variants with increasing context length and segment compelxity: Easy (2 topics, ABAB pattern, 1.5K tokens), Medium (2 topics, AABBAABB, 2.5K tokens), Hard (3 topics, ABCABC, 2K tokens), and Very Hard (3 topics, AAABBBCCC, 4K tokens). The Very Hard variant is particularly challenging because it requires maintaining compressed representations of Topics A and B throughout the entire span of Topic C before they become relevant again.

4.2 Setting

On the WikiSalad dataset, we use the model Qwen3-4B-Instruct-2507 and run the benchmarks on the baseline model, as well as lined-only, clustered-only, and mixed attention modes.

Our clustered attention was configured as follows: we used a fixed segmenter creating pages of length 16, and our attention-weighed compressor. We refined using Top-k with $k = 3$, as our ablations (Section 5) showed that even minimal amounts of refinement significantly improves model perplexity. Our lined attention configuration was similar, with a budget of keeping roughly $1/8$ of the heavy hitters with respect to the prompt length. This ensured an apples-to-apples comparison, as the rate of compression is effectively the same, roughly $1/8\times$ the number of original tokens being utilized between the two methods. Both methods also were configured so that a tail of at least 128 tokens were kept raw for accurate "local" attention.

For the mixed attention mode, the first 8 and last 8 are set to lined attention, with the remaining 20 middle layers set to clustered attention. All experiments are conducted on NVIDIA A6000 GPUs. Performance is measured using Exact Match (EM) to measure answer accuracy.

4.3 Results on WikiSalad

The results of our experiment can be found in Table 1. With the baseline model, we can see that our labels of difficulty are roughly aligned with model accuracy, with the easy dataset questions being correctly answered at 75.0%, and the very hard quetions only being correctly answered at 52.8%.

Dataset	Baseline	Lined (H_2O) Only	Clustered Only	Mixed
Easy	75.0	30.0	57.5	40.0
Medium	50.0	23.8	45.0	26.2
Hard	55.0	18.3	41.7	23.3
Very Hard	52.8	20.0	37.2	24.4

Table 1: Exact Match (EM) on WikiSalad. Baseline uses full attention.

With lined attention, we observed a significant drop in answer quality, with a drop of 45% on the easy dataset and a 32.8% on the very hard dataset. On the other hand, the clustered-only mode demonstrates better results across all dataset difficulties than lined-only but falls short of the baseline.

The mixed configuration performs between lined-only and clustered-only. This result is surprising to us, as we hypothesized that aligning the lined attention patterns with H_2O would enable more aligned sparse attention patterns, yielding the best approximation of original attention. However, this did not end up being the case, and ended up being an interpolation of the two modes.

4.4 Runtime Analysis

We also measured the runtime of our model compared to the baseline model. Taking the Qwen3-0.6B model, we measure the average latency of the attention mechanism starting at position length 2048, for 64 decode steps. While we planned to profile runtime at longer lengths (over 10K tokens), we ran into OOM issues on a single A6000 reaching 8196 tokens, as well as issues with inference on multiple GPUs. We used a refinement rule of Top-k with $k = 3$, so a constant number of refinements are done per step. Every 16 decode steps, we also run the segmenter and compressor.

Figure 3 illustrates the breakdown of the latency: the clustered attention mechanism, by itself, indeed is faster than the raw attention, which makes sense because there are less computations to be done. However, the runtime of our clustered attention implement is *dominated* by overhead from the refinement operations, which includes masking and new tensor creations. The segmenter and compressor operations also have heavy overhead, as this latency in the figure shown is also only $1/16$ of the actual operation.

Ultimately, improving the runtime analysis is future work. One way we can improve the overhead of the segmenter and compressor operations is to make them *asynchronous*, offloading these operations on a different GPU, so the main decoding loop does not have to wait after the attention computation for new pages to be created.

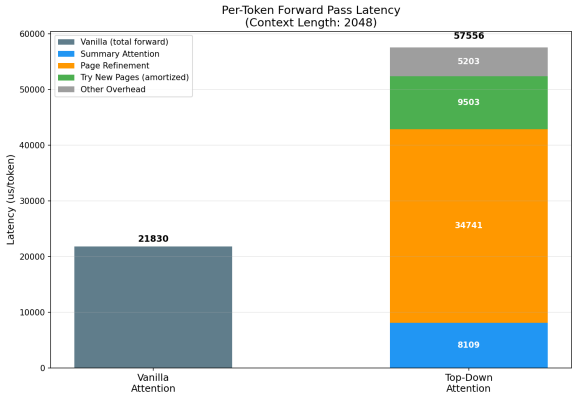

Figure 3: **Latency Breakdown.** The breakdown of the latency numbers in the clustered attention operations, compared to normal attention.

5 Ablations

In addition to the main experiments, we also experimented on different hyperparameters within the clustered attention operation.

Compressor Types. We analyzed different types of compressors, as well as how much refinement mattered to the model's performance. Using the Top-Frac rule, we measured perplexity on the first example of the pg19 [2] dataset for 512 tokens on different compressors, including mean, attention-weighted, and random compressors as mentioned in section 3.

In Figure 4, the x-axis is the proportion of refinement; 0 indicates zero refinement, while 1 indicates full refinement (roughly that of raw attention). We see that attention weighted does the best as refinement becomes less and less frequent, and at 0 does not blow up compared to the others, demonstrating the superiority of this compressor type.

Perplexity by Attention Mode. We also compare perplexity between the different attention modes: lined-only, clustered-only, and mixed-only (using the same configuration in the experimental setup

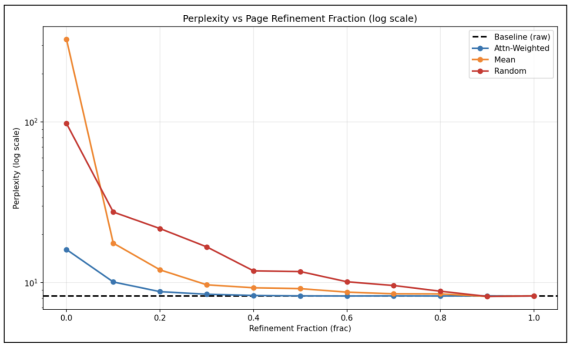

Figure 4: **Impact of Refinement Fraction on Perplexity.** Lower refinement fractions lead to higher perplexity, but attention-weighted compression mitigates this loss.

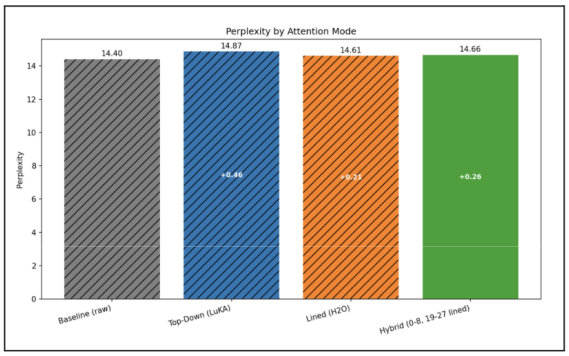

Figure 5: **Perplexity Between Attention Modes**. H_2O, Clustered, and Mixed all demonstrate comparable performance to the baseline model.

in Section 4), using the first example in the pg19 dataset over 1024 tokens. The results are shown in Figure 5; all sparse variants exhibit perplexity close to the baseline (< 1 difference), demonstrating that these methods are preserving quality.

Importance of Summary Values. We created modified compressors that also output $V = 0$, which means the summary contributions are essentially null, mimic-ing eviction behavior. Running the same experiment above, are results are shown in Figure 6. Setting summary values to zero increases perplexity significantly at low refinement fractions, proving that adding the constructed summary values still provide useful context.

6 Limitations and Future Work

While LuKA demonstrates promising results relative to pure H_2O, several limitations remain.

Runtime Overhead. The primary limitation of LuKA is runtime efficiency. Although clustered attention theoretically reduces cost of attention by

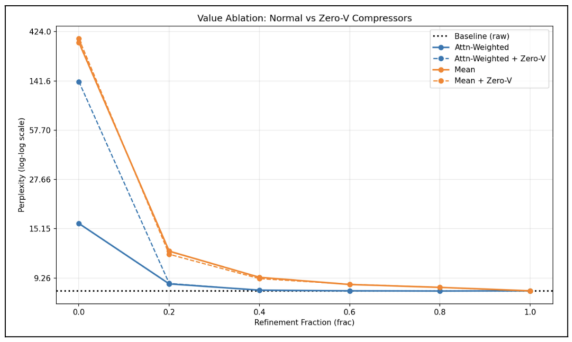

Figure 6: **Ablation of Summary Values.** Zeroing out summary values leads to a sharp increase in perplexity, highlighting their importance.

operating over a compressed cover at long horizons, our current implementation incurs substantial overhead from refinement logic, masking, to additional tensor management. As shown in Section 4.3, these overheads dominate end-to-end latency, offsetting the computational savings of reduced attention. Segmenter and compressor operations further contribute nontrivial cost, particularly because they are executed synchronously within the decoding loop. As a result, LuKA in its present form should be viewed as a proof-of-concept rather than a drop-in replacement for optimized attention kernels.

Memory Trade-offs. LuKA prioritizes information retention over aggressive memory reduction. Maintaining summary values, hierarchical metadata, and refinement state introduces additional memory overhead compared to strict eviction-based methods. As such, LuKA trades peak memory efficiency for reversibility and fidelity, which may not be suitable for all deployment settings.

Exploration of Other Models. Due to time constraints and complexity of implementation of the infrastructure, our compressor, segmenter, and refinement rules were rather similar. While we did explore trained compressor models, they ultimately proved to be difficult to generalize correctly, and rule-based methods seemed to peform better overall.

7 Conclusion

We introduced LuKA, a structured KV-cache design for efficient long-context inference that supports clustered, lined, and mixed attention mechanisms. LuKA reduces attention computation by compressing older context into summary represen-

tations, preserving globally important tokens, and selectively refining summaries when fine-grained information is needed. Across experiments on the WikiSalad benchmark, clustered attention consistently outperformed lined and mixed attention, achieving better accuracy yet its efficiency underperforms in both memory and compute. Our ablation studies show that refinement frequency is critical, summary values encode meaningful information beyond simple eviction, and attention-weighted compression is robust to design choices such as bias terms. While LuKA does not fully match full-attention accuracy on the hardest tasks, it preserves token-level predictive quality while substantially reducing the effective attention span. Together, these results suggest that hierarchical compression with selective refinement is a promising direction for scalable long-context inference, and motivate future work on adaptive segmentation and token recovery mechanisms.

8 Contributions

Rolando: Designed and implemented the WikiSalad benchmark dataset, including the interlacing methodology for creating multi-topic contexts from SQuAD. Developed the dataset generation pipeline with four difficulty variants and the evaluation metrics framework. Contributed to experimental design, model architectural design, evaluation infrastructure, and report writing

Anthony: Implemented the infrastructure for LuKA, interfacing the Qwen models with our custom Hugging Face layers through monkey-patching. Implemented the algorithm for clustered attention. Experimented with trained compressors (which ultimately were found to be not very effective), and ran perplexity and runtime experiments.

Krishna: Implemented the initial lined attention mechanism based on H_2O, including the heavy-hitter cache, attention score accumulation, grid selection, and cover construction. Assisted with evaluation experiments, generated Exact Match and throughput visualizations, and contributed to writing and editing the report.

Gerardo: Implemented lined attention and developed the evaluation framework for different model configurations. Ran evaluations across baseline, H_2O, Clustered, and Mixed attention on different difficulty WikiSalad datasets. contributed to writing the report and overall formatting.

References

[1] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *arXiv*.

[2] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. 2019. Compressive transformers for long-range sequence modelling. *arXiv preprint*.

[3] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *Preprint*, arXiv:1606.05250.

[4] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv*.

[5] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. *arXiv*.

[6] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. 2025. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv*.

[7] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H_2o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv*.