Sequence can Secretly Tell You What to Discard

Jincheng Dai
ZheJiang University
jincheng_dai@zju.edu.cn

Zhuowei Huang
ZheJiang University
zw_huang@zju.edu.cn

Haiyun Jiang*
Tencent AI Lab
haiyunjiang@tencent.com

Chen Chen
Tencent AI Lab
chenzchen@tencent.com

Deng Cai
Tencent AI Lab
jcykcai@tencent.com

Wei Bi
Tencent AI Lab
victoriabi@tencent.com

Shuming Shi
Tencent AI Lab
shumingshi@tencent.com

Abstract

Large Language Models (LLMs), despite their impressive performance on a wide range of tasks, require significant GPU memory and consume substantial computational resources. In addition to model weights, the memory occupied by KV cache increases linearly with sequence length, becoming a main bottleneck for inference. In this paper, we introduce a novel approach for optimizing the KV cache which significantly reduces its memory footprint. Through a comprehensive investigation, we find that on LLaMA2 series models, (*i*) the similarity between adjacent tokens' query vectors is remarkably high, and (*ii*) current query's attention calculation can rely solely on the attention information of a small portion of the preceding queries. Based on these observations, we propose CORM, a KV cache eviction policy that dynamically retains important key-value pairs for inference without finetuning the model. We validate that CORM reduces the inference memory usage of KV cache by up to 70% without noticeable performance degradation across six tasks in LongBench.

1 Introduction

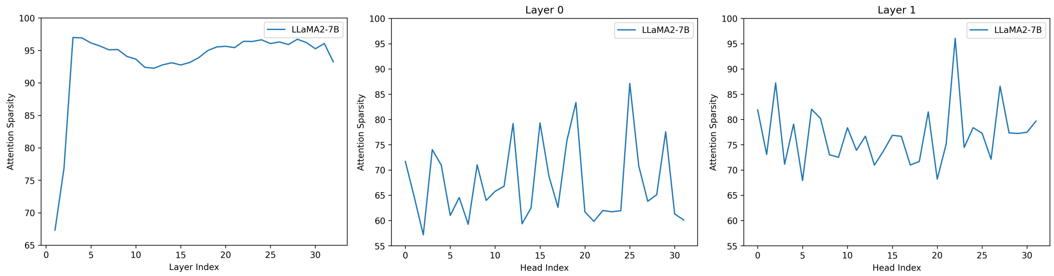
Large language models (LLMs) have demonstrated impressive proficiency in a wide range of natural language processing tasks such as question answering, summarization and multi-turn dialogues [1–3]. Considering substantial cost of deploying LLMs introduced by tremendous model size and quadratic cost of attention layer, many works focused on model compression and memory-efficient attention techniques [4–7]. However, the size of KV cache, which stores previous tokens' key and value states to avoid re-computation, scaling linearly with sequence length during generation, also incurs significant overhead. For instance, even a 7 billion-parameter model with batch size of 128 and sequence length of 4096 results in 256GB of KV cache, far exceeds memory consumed by model itself which is only 14GB. A natural idea is to discard some less informative KV cache to reduce memory consumption. The challenge lies in finding a balance between discarding as much as possible while still maintaining model performance.

Despite multi-query attention [8] and grouped-query attention [9] can reduce the size of KV cache by reducing attention heads, it needs re-training to recover performance of original model. Recent works

*Corresponding Author

Preprint. In progress.

[10–14] have investigated implementing KV cache using specific eviction policy, that determines which key-value states should be evicted from KV cache. These methods aim to compress KV cache to a pre-defined budget size, thereby reducing memory and computational overhead. However, they save same number of key-value pairs for all attention heads and layers, ignoring that the number of keys playing an important role may vary across different attention heads and layers [15].



(a) (b)

Figure 1: Attention sparsity of LLaMA2-7B. (a) Layer-wise attention sparsity. (b) Head-wise attention sparsity of layer 0 and layer 1.

Intuitively, if important information in the KV cache exceeds the predetermined budget size, the performance of the model is likely to decline as it unavoidably evicts some crucial information. Our preliminary exploration also reveals that different attention layers and heads show different sparsities as shown in Figure 1. First, we observe that bottom layers of the model are relatively dense[2], while the remaining attention layers exhibit significant sparsity. Second, even within the same layer, different heads can exhibit obvious differences in sparsity levels. These properties suggest that we need to treat different layers and heads differently, rather than using the same budget size for all of them. In addition, we prove that completely similar queries have similar concerns about keys, and observe that recent query vectors are quite similar on LLaMA2 series models so current query can directly use recent query attention messages during generation.

Based on the above insights, we first define the generation process of LLMs with a budget-unrestricted KV cache in Section 3. Then we propose **C**ache **O**ptimization with **R**ecent **M**essage (CORM), a framework that exploits recent query attention information for KV cache optimization and token generation of LLMs. Specifically,

- In Section 3, we explore the similarity between query vectors of all tokens within same sequence, revealing that recent query vectors are highly similar, which implies that (*i*) keys that are important for recent queries might be also important for the current query; and (*ii*) removing key-value pairs that appear to be less informative for recent queries can greatly preserve the performance of the model.
- In Section 4, we present a simple method which dynamically evicts minor key-value pairs determined by recent tokens' attention information.

We conduct extensive experiments on LLaMA2-7B-Chat, considering its popularity and wide usage, to evaluate CORM across 6 tasks from LongBench [16] containing question answering, summarization, code completion, etc. Experiments show that even without explicitly setting a budget size, our method is still possible to achieve a high compression rate. Our method achieves better performance compared to StreamingLLM [10], Scissorhands [11] and H_2O [12] with over 70% KV cache reduction rate and can even come close to fully restoring the performance of the model.

2 Related Work

Attention Let $x \in \mathbb{R}^{n \times d}$ denote the input embeddings from a sequence of n feature vectors of dimension d. The multi-head self-attention [17], as a core module of Transformer model, facilitates

[2] Let t denote sequence length, we count the proportion of keys which attention score larger than average score $\frac{1}{t}$ and denote it as r. The larger r is, the sparser the layer is.

contextual information interaction within each head in the following manner:

$$
\begin{aligned}
Q &= xW_q, \\
K &= xW_k, \\
V &= xW_v, \\
\mathit{Attention}(x) &= \mathit{softmax}(\frac{QK^T}{\sqrt{d_h}}) \times V
\end{aligned} \tag{1}
$$

Q, K, V represent the query, key, and value matrices, which are obtained by linearly mapping x using weight matrices W_q, W_k, and $W_v \in \mathbb{R}^{d \times d_h}$, respectively. d_h is the dimension of each individual head.

KV Cache According to autoregressive paradigm, transformer decoder model predicts future tokens based on both previous and current tokens. Recalculating the key-value pairs for previous tokens at each decoding step is clearly an inefficient strategy. A common practice is to retain the key-value pairs of previous tokens for subsequent reuse. Thus, the consumption of KV cache becomes linearly correlated with the length of input sequence. When dealing with long contexts, however, the use of such a space-time trade-off approach may still pose challenges.

Training Policies The advent of multi-query attention (MQA) [8] is to address the influence of attention heads on KV cache within multi-head attention (MHA) mechanism. It facilitates the sharing of the same set of keys and values among different heads to alleviate cache pressure. Grouped-query attention (GQA) [9] represents a trade-off between MHA and MQA, achieving key-value sharing within each group through mean-pooling-based uptraining. Both methods require additional training to restore model performance due to the inability to directly convert.

Training-free Policies During generation, sequence length is the primary factor of cache pressure. Recent methods aim to balance model efficiency and inference cost without extra training and architectural changes. StreamingLLM [10] keeps attention sink token and recent tokens throughout decoding process to align with the training window. Scissorhands [11] maintains pivotal tokens and recent tokens based on the persistence of importance hypothesis. H_2O [12] utilizes accumulated attention score to maintain heavy hitters and recent tokens. TOVA [13] removes tokens with the lowest current attention score from the fixed cache at each decoding step. RoCo [14] retains tokens in the fixed cache based on high mean cumulative attention scores and top r standard deviations. Aforementioned methods consistently operate on a fixed cache, ignoring that the number of tokens playing an important role may vary across different attention heads and layers.

3 Observations

We first demonstrate *the existence of attention sparsity in LLMs* in Section 3.1, then discuss the phenomenon that *similar queries have similar attention concerns for keys* in Section 3.2. In Section 3.3, we show an intriguing observation that *current query is most similar to recent queries*.

3.1 Attention sparsity in LLMs

We first explore the sparsity in attention layers of LLMs, which provides an effective guarantee for us to reduce KV cache size. Specifically, we use proportion of important keys to represent attention sparsity. Let $q_t \in \mathbb{R}^{1\times d}$ denote the query state vector at step t, $k_i \in \mathbb{R}^{1\times d}$ denote the key state vector at step i $(1 \leq i \leq t)$, where d is hidden dimension (for the sake of simplicity, we only consider a single head here). The normalized attention score of q_t for k_i is computed as:

$$
\alpha_{t,i} = \frac{\exp(q_t k_i^T/\sqrt{d})}{\sum_{j=1}^{t}\exp(q_t k_j^T/\sqrt{d})}. \tag{2}
$$

Definition 3.1 (Important Key) We define a key k_i is considered important in step t, if and only if $\alpha_{t,i} \geq \frac{1}{t}$, otherwise it is considered minor.

We conduct zero-shot inference with LLaMA2-7B model on the test set of PG-19 [18]. We plot the layer-wise and head-wise sparsity within attention blocks, the results are presented in Figure 1. It reveals that bottom layers are relatively dense, while other layers are highly sparse with over 90% sparsity. This makes it possible to do attention computation on only small part of KV cache during generation.

3.2 Similar queries have similar concerns for keys

The previous section reveals the existence of attention sparsity in LLMs, which provides an opportunity to reduce KV cache size while maintaining performance. In this section we give a theoretical analysis that *similar queries have similar concerns for keys* for eviction policy design.

Consider the i-th and j-th query state vectors q_i and q_j in a sequence of token length T ($i < j \leq T$). Their cosine similarity can be computed as:

$$\mathit{cosine\_similarity}(q_i, q_j) = \frac{q_i q_j^T}{\|q_i\| \cdot \|q_j\|}. \tag{3}$$

Consider all key states k_1, k_2, ..., k_{i-1} before i-th key. Assume that $\mathit{cosine\_similarity}(q_i, q_j) = 1$, then $q_i = m \cdot q_j$ with $m \in \mathbb{R}^+$. The attention weight[3] of q_i to the previous $i-1$ keys can be represented as:

$$\mathit{attention\_weight} = \frac{1}{\sqrt{d}}(q_i k_1^T, q_i k_2^T, ..., q_i k_{i-1}^T) = \frac{m}{\sqrt{d}} \cdot (q_j k_1^T, q_j k_2^T, ..., q_j k_{i-1}^T). \tag{4}$$

Note that m is a positive number that does not affect the relative order of the attention weights. For example, for q_i, if $q_i k_1^T > q_i k_2^T$, there must be $q_j k_1^T > q_j k_2^T$ for q_j. This means if a key is important to q_i, it is also important to q_j, though the degree of importance may vary due to the softmax function.

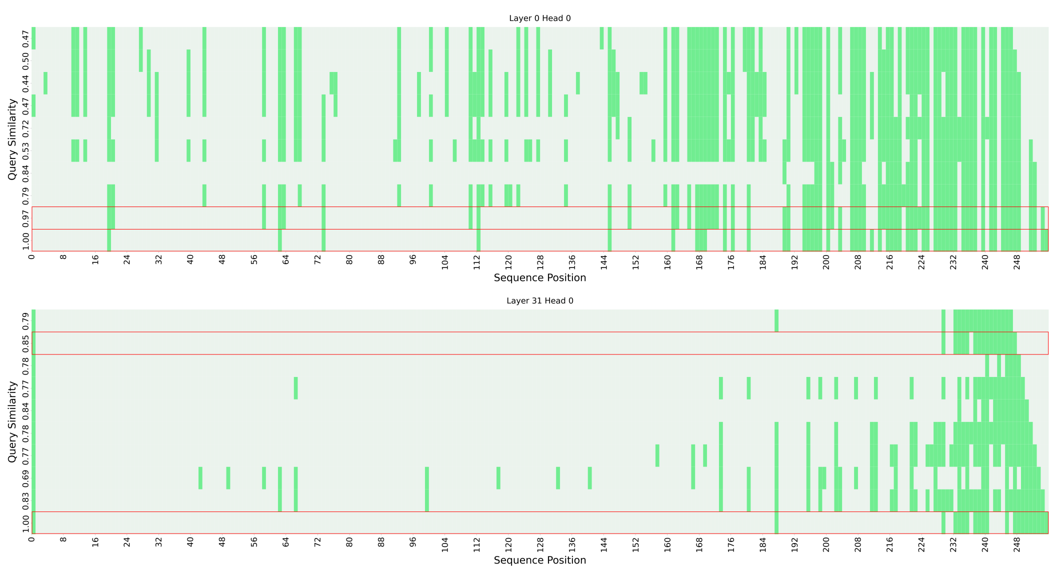


Figure 2: Similar queries have similar concerns for keys. We plot the attention map from two different layers in a sentence. We discretize the attention score and those important keys are shown in bright green. Each attention map has two red borders, the bottom border shows important keys that current query actually focuses on, while another border shows important keys that the most similar query focuses on.

Although it's nearly impossible that $\mathit{cosine\_similarity}(q_i, q_j) = 1$ in real situation, we can make the hypothesis that two similar queries may have similar concerns for keys. To validate this hypothesis, we provide two attention maps of a sentence randomly drawn from PG-19 using LLaMA2-7B, as

[3] attention weight is unnormalized attention score

shown in Figure 2. Important keys are marked with bright green, more plots are available in Appendix A.1. We observe that the hypothesis is true, and similar queries exhibit similar concerns for important keys. At the same time, important keys only account for a small proportion especially in deeper attention layers, which is consistent with the finding that deeper layers are sparser in previous section.

3.3 Similarity exploration of query vectors

We have validated *two similar queries have similar concerns for keys* in Section 3.2, we also need to validate that at each step we can find a previous query state that is similar enough to current query state in same layer and same head. To check this, we visualize cosine similarity of query vectors within same sequence as shown in Figure 3, more plots are available in Appendix A.2. We observe an intriguing phenomenon that many images show clear oblique color segmentation, with the top oblique block closest to dark red which means current query is most similar to recent queries.

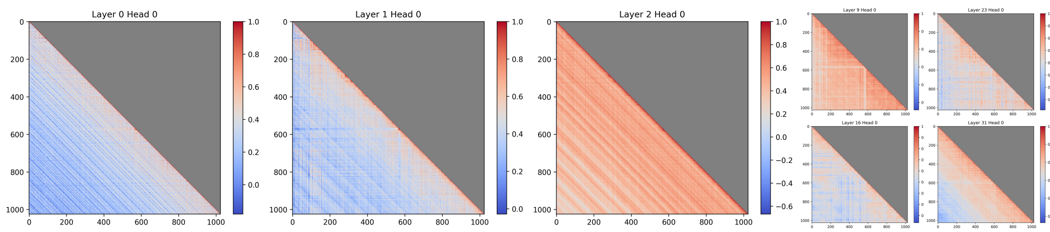


Figure 3: Visualization of query vectors' cosine similarity over one sentence with a length of 1024. The i-th row of the map represents cosine similarity of the i-th query to all previous queries. The plot reveals that in most cases current query is most similar to recent queries.

Through above observations, we see an opportunity to design a KV cache eviction policy based on query similarity that preserves the LLM generation performance.

4 Cache Optimization with Recent Message

In this section, we present CORM, a method reduces the KV cache memory based on recent query attention information without any fine-tuning process. In Section 4.1, we derive that current query can directly use recent query attention messages during generation. In Section 4.2, we present CORM eviction policy and describe how it works during generation.

4.1 Generate based on recent query attention messages

Consider observations in Section 3, intuitively, we can directly store all queries and their attention information for future reference. At each generation step, use current query to find the most similar one from previous queries, and use its saved attention information to calculate solely on important keys. However, this approach incurs a significant cost. First, storing all queries results in a substantial increase in memory overhead. Second, the requirement of performing similarity calculations at each step adds to the computational overhead.

Since in most cases current query is most similar to recent queries as described in Section 3.3, we can just use recent query attention messages. And from Figure 2 we can also observe that only a small proportion of keys are considered important by recent queries. Therefore even if we save all the keys that are considered important in previous steps, we can save a lot of memory.

4.2 Eviction algorithm via recent message

We have shown recent query attention information is enough for cache optimization in Section 4.1. In the following, we formally define this algorithm and introduce how to integrate it into LLM generation directly.

Definition 4.1 (Long-term Minor Key) A key k_i is considered as long-term minor key only if it is considered minor in all recent r steps (from $t - r + 1$ to t).

Approach CORM will have a recent window of size w to record the information of recent w queries, and will always keep recent r keys unremoved to prevent them from being discarded prematurely due to insufficient observations. During generation, k_i, v_i will be discarded once k_i is regarded as long-term minor key. For better explanation we present pytorch code[4] of main algorithm in Algorithm 1. Intuitively, when w is larger, more keys and values will be saved, the compression rate will be smaller and performance will be better; Conversely, when w is smaller, fewer keys and values will be saved, the compression rate will be larger and performance will be worse. So there's a tradeoff between performance and compress rate.

Memory Overhead Analysis In order to reduce memory overhead of KV cache, an extra memory overhead is introduced by recent information cache. We need to store recent query messages which increase memory overhead. However, these overheads are far less than compressed KV cache, one can use a small portion of memory to avoid maintaining full KV cache memory without obvious performance degradation. On the other hand, the compression rate will increase as the sequence length increases as shown in Figure 4, resulting in a lower memory overhead for this component in comparison.

Algorithm 1 Single-head KV cache eviction with CORM (unbatched)

```
def corm_eviction(keys, values, message, attn_score, w, r, t):
"""
Args:
    keys: previous key states, a tensor with shape [l, d]
    values: previous value states, a tensor with shape [l, d]
    message: attention message, a tensor with shape of [m, l-1]
    attn_score: current steps attention score, a tensor with shape of [1, l]
    w: window size, a scalar
    r: recent size, a scalar
    t: current step, a scalar
Returns:
    updated_keys: updated keys
    updated_values: updated values
    updated_message: updated message
"""
m = message.shape[0]

# update attention message
message = torch.cat([message, torch.zeros(m, 1)], dim=1)          ▷ pad to [m, l]
cur_message = attn_score >= 1 / t
message = torch.cat([message, cur_message], dim=1)[-w:, :]

if message.shape[0] < w:
    return keys, values, message
else:
    # determine the key-value pairs that necessitate discarding
    decision = message.any(dim=0)
    decision[-r:] = True                    ▷ always keep recent r tokens unremoved
    indices = torch.nonzero(decision).squeeze()

    keys = keys[indices, :]
    values = values[indices, :]
    return keys, values, message
```

[4]For the sake of brevity, the code snippet only demonstrates single-head eviction operation, while in the actual implementation, it will be performed on each head at every layer.

5 Empirical Evaluation

In this section, we present the results that demonstrate CORM can reduce up to 70% of the memory footprint of KV Cache without accuracy degradation on LLaMA2-7B-Chat.

Dataset To broadly validate feasibility of our method on real-world use cases, we choose LongBench [16] as our evaluation benchmark, which contains a wide range of long-text tasks such as question answering [19–24], summarization [25–28], few-shot learning [29–32], synthetic task and code completion [33, 34]. Here we do not consider short text tasks, because even full cache doesn't have any bottlenecks.

Models Since sequence length is the main factor in the continuous growth of KV Cache, we employ LLaMA2-7B-Chat [2] for 4K test considering its wide usage.

Baselines Since CORM reduces KV cache without need for training, we consider several similar approaches as our baselines: StreamLLM [10], Scissorhands [11] and H_2O [12]. In addition, the full KV cache is also considered as strong baseline to measure the performance loss of other methods.

Setting All baselines can be regarded as fixed budget size KV cache compression, however CORM is a dynamic compression method. Since we find that CORM has similar compression rates for various task texts with the same sequence length. For fair comparison, we plot the relationship between model compression rate and sequence length using texts randomly sampled from PG19 [18] as shown in Figure 4.

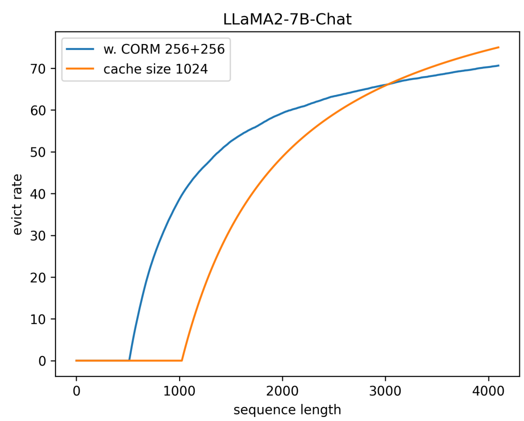


Figure 4: Relationship between compression ratio and sequence length. Plots show that compression rate with CORM "256+256" and budget=1024 are close for LLaMA2-7B-Chat.

Main Results We evaluate LLaMA2-7B-Chat for 4K length text. Results are summarized in Table 1 & 2 for LLaMA2-7B-Chat. The following observations can be drawn: (1) CORM consistently outperforms previous methods at the same compression rate across a wide range of tasks. (2) Meanwhile, with over 70% KV cache reduction, CORM achieves comparable performance as the model with full KV cache and even surpass it on some tasks, we speculate it's because there's some noise in full KV cache that affects model output and our method can eliminate this noise to a certain extent by discarding some KV cache.

5.1 Budget unnecessity: is unbudgeted better?

We primarily focus on the effectiveness of not setting a budget versus setting a fixed budget. Note that since we use same window size and recent size as Scissorhands in the experiment, it can be regarded a natural ablation experiment. And Table 1 & 2 have shown that, at the similar compression rate, CORM is much better than Scissorhands in most tasks, and performance of other tasks is close. This verifies that different transformer layers and heads should be treated differently rather than setting a same fixed budget size.

Table 1: Results (%) on single-doc QA, multi-doc QA and summarization tasks. "Full" refers to LLaMA2-7B-Chat utilizing full KV Cache, "StreamLLM" is configured with 4+1020, "Scissorhands" is configured with 768+256 where window size=256, "H_2O" is configured with 768+256, "CORM" is configured with 256+256 for fair comparison. For the sake of brevity we use ID to denote dataset here, mapping from ID to dataset can be found in Appendix B

.

Method	Single-Doc QA				Multi-Doc QA				Summarization			
	1-1	1-2	1-3	1-4	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4
Full	19.0	22.1	36.7	11.8	27.8	31.5	8.3	6.8	26.8	20.7	26.2	0.2
StreamLLM	13.2	15.4	27.2	6.5	24.2	25.4	5.3	4.4	21.6	19.8	24.4	0.1
Scissorhands	16.6	18.7	32.4	9.9	26.3	32.1	8.9	5.7	22.1	20.7	25.4	0.2
H_2O	17.9	19.5	34.9	11.5	27.5	29.7	7.5	7.1	24.5	21.0	25.8	0.2
CORM	18.9	22.2	38.6	12.0	27.6	31.6	8.4	7.1	26.4	21.0	25.8	0.2

Table 2: Results (%) on few-shot learning, synthetic, and code tasks. "Overall" is computed by the macro-average over major task categories. This is computed on English (EN) tasks, Chinese (ZH) tasks, and all (All) tasks, code tasks are included in both languages.

Method	Few-shot Learning				Synthetic			Code		Overall		
	4-1	4-2	4-3	4-4	5-1	5-2	5-3	6-1	6-2	EN	ZN	All
Full	64.0	83.3	41.4	17.3	2.9	7.8	10.0	58.3	52.2	32.8	16.9	28.9
StreamLLM	61.0	82.9	39.1	14.5	1.8	4.7	6.5	57.6	50.0	29.5	14.3	25.7
Scissorhands	52.5	83.6	40.7	17.0	3.1	6.5	7.7	56.8	52.1	31.0	15.8	27.2
H_2O	63.0	81.5	39.9	17.0	2.8	7.0	7.3	57.8	52.3	31.8	16.4	28.0
CORM	64.0	83.5	41.3	17.3	2.9	9.0	9.1	58.3	52.0	32.9	16.8	28.9

6 Conclusion

In this paper, we investigate a critical memory bottleneck in LLM deployment, KV cache. Inspired by similar queries have similar concerns for keys and recent queries are similar enough, we propose CORM, an unbudgeted KV cache eviction policy for significantly reducing its memory footprint, by reusing recent query attention messages. Through extensive evaluations, we demonstrate that CORM can reduce the inference memory usage of the KV cache by up to 70% without noticeable performance degradation across a variety of tasks.

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[3] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[4] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.

[5] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

[6] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[7] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

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

[9] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

[10] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[11] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024.

[12] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[13] Matanel Oren, Michael Hassid, Yossi Adi, and Roy Schwartz. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*, 2024.

[14] Siyu Ren and Kenny Q Zhu. On the efficacy of eviction policy for key-value constrained generative language model inference. *arXiv preprint arXiv:2402.06262*, 2024.

[15] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

[16] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[18] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

[19] Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

[20] Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021.

[21] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

[22] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020.

[23] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

[24] Wei He, Kai Liu, Jing Liu, Yajuan Lyu, Shiqi Zhao, Xinyan Xiao, Yuan Liu, Yizhong Wang, Hua Wu, Qiaoqiao She, et al. Dureader: a chinese machine reading comprehension dataset from real-world applications. *arXiv preprint arXiv:1711.05073*, 2017.

[25] Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. *arXiv preprint arXiv:2104.02112*, 2021.

[26] Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, et al. Qmsum: A new benchmark for query-based multi-domain meeting summarization. *arXiv preprint arXiv:2104.05938*, 2021.

[27] Alexander R Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749*, 2019.

[28] Han Wu, Mingjie Zhan, Haochen Tan, Zhaohui Hou, Ding Liang, and Linqi Song. Vcsum: A versatile chinese meeting summarization dataset. *arXiv preprint arXiv:2305.05280*, 2023.

[29] Xin Li and Dan Roth. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002.

[30] Task definition for large scale text categorization at nlpcc 2014. `http://tcci.ccf.org.cn/conference/2014/dldoc/evatask6.pdf`, 2014.

[31] Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *arXiv preprint arXiv:1911.12237*, 2019.

[32] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

[33] Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning*, pages 12098–12107. PMLR, 2023.

[34] Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023.

A More Plots

A.1 Similar queries share most of important tokens

We provide the attention map similar to Figure 2 but from different heads on the same text in Figure 5, Figure 6. Plots from a different layer on the same text are shown in Figure 7.

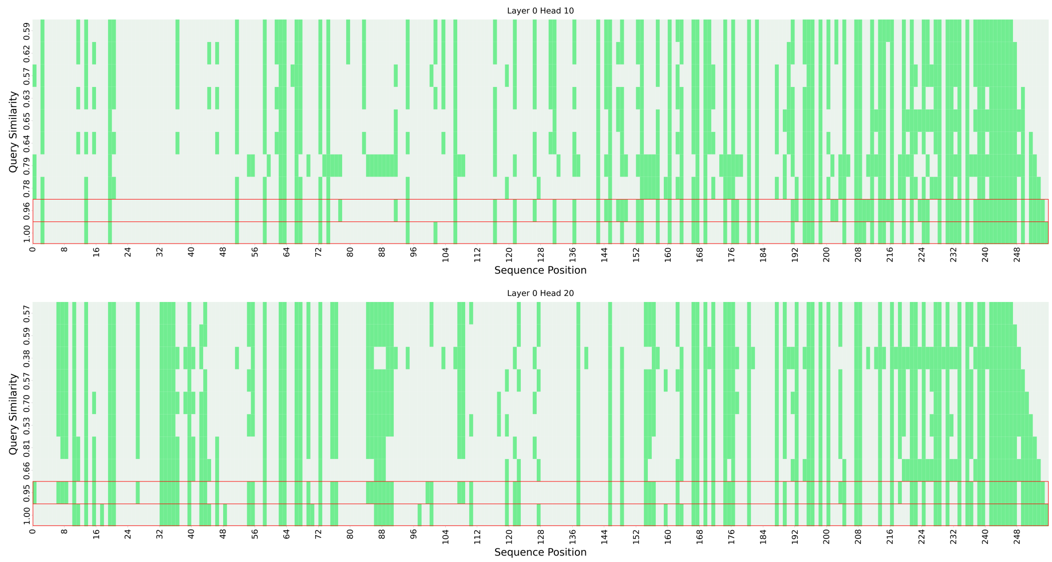


Figure 5: Attention Map at Layer 0, Head 10, 20

Figure 6: Attention Map at Layer 31, Head 10, 20

A.2 Relationship of query vectors

We provide the query vectors' cosine similarity visualization similar to Figure 3 but from different layers and heads on the same text in Figure 8.

Figure 7: Attention Map at Layer 5, 15, 25, Head 0

B Task Mapping

An overview of the dataset statistics and mapping from ID to dataset in LongBench is shown in Figure 9.

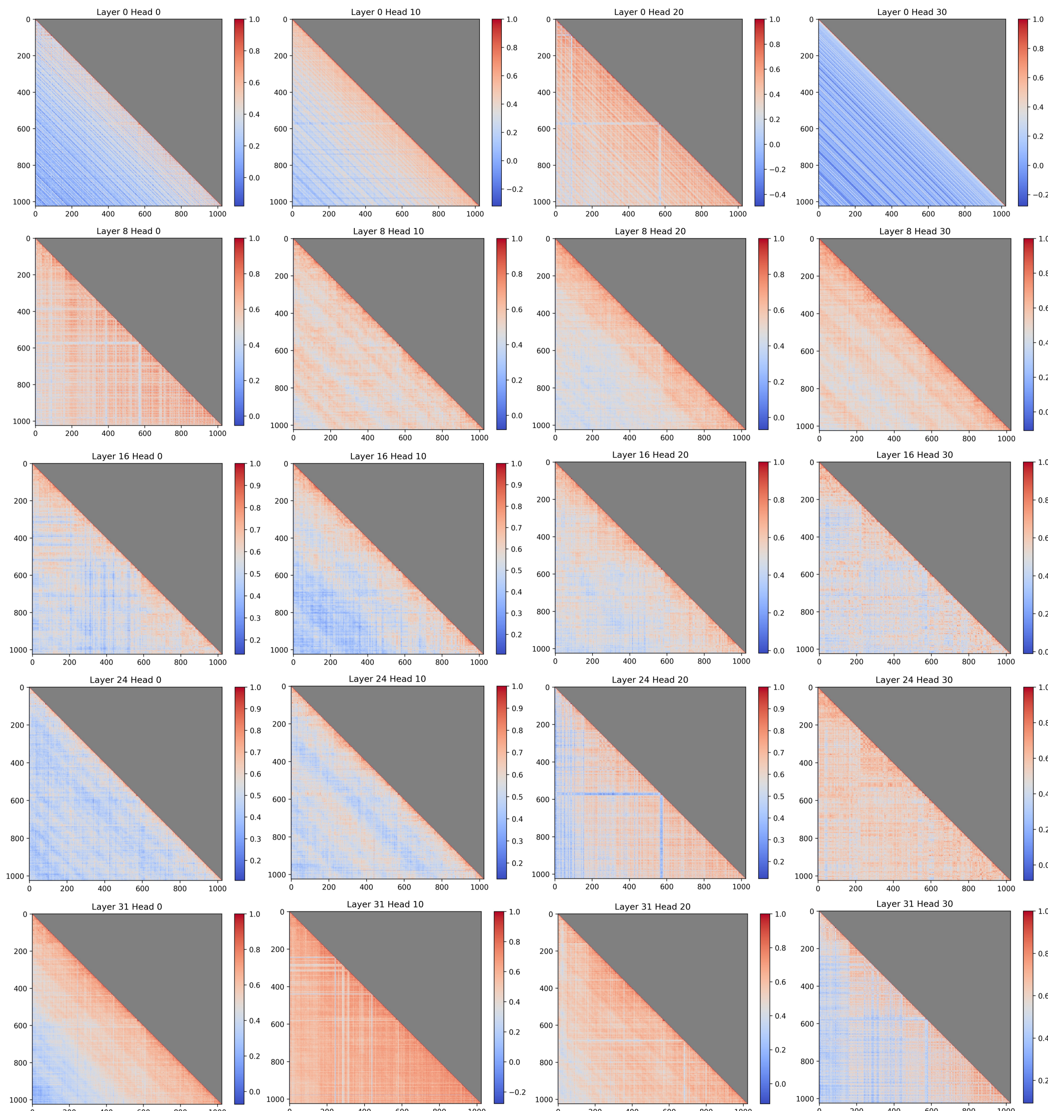


Figure 8: Visualization of query vectors' cosine similarity across different layers and heads

Dataset	ID	Source	Avg len	Metric	Language	#data
Single-Document QA						
NarrativeQA	1-1	Literature, Film	18,409	F1	English	200
Qasper	1-2	Science	3,619	F1	English	200
MultiFieldQA-en	1-3	Multi-field	4,559	F1	English	150
MultiFieldQA-zh	1-4	Multi-field	6,701	F1	Chinese	200
Multi-Document QA						
HotpotQA	2-1	Wikipedia	9,151	F1	English	200
2WikiMultihopQA	2-2	Wikipedia	4,887	F1	English	200
MuSiQue	2-3	Wikipedia	11,214	F1	English	200
DuReader	2-4	Baidu Search	15,768	Rouge-L	Chinese	200
Summarization						
GovReport	3-1	Government report	8,734	Rouge-L	English	200
QMSum	3-2	Meeting	10,614	Rouge-L	English	200
MultiNews	3-3	News	2,113	Rouge-L	English	200
VCSUM	3-4	Meeting	15,380	Rouge-L	Chinese	200
Few-shot Learning						
TREC	4-1	Web question	5,177	Accuracy (CLS)	English	200
TriviaQA	4-2	Wikipedia, Web	8,209	F1	English	200
SAMSum	4-3	Dialogue	6,258	Rouge-L	English	200
LSHT	4-4	News	22,337	Accuracy (CLS)	Chinese	200
Synthetic Task						
PassageCount	5-1	Wikipedia	11,141	Accuracy (EM)	English	200
PassageRetrieval-en	5-2	Wikipedia	9,289	Accuracy (EM)	English	200
PassageRetrieval-zh	5-3	C4 Dataset	6,745	Accuracy (EM)	Chinese	200
Code Completion						
LCC	6-1	Github	1,235	Edit Sim	Python/C#/Java	500
RepoBench-P	6-2	Github repository	4,206	Edit Sim	Python/Java	500

Figure 9: An overview of the dataset statistics in LongBench