

# LSH TELLS YOU WHAT TO DISCARD: AN ADAPTIVE LOCALITY-SENSITIVE STRATEGY FOR KV CACHE COMPRESSION

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

Transformer-based large language models (LLMs) use the key-value (KV) cache to significantly accelerate inference by storing the key and value embeddings of past tokens. However, this cache consumes significant GPU memory. In this work, we introduce LSH-E, an algorithm that uses locality-sensitive hashing (LSH) to compress the KV cache. LSH-E quickly locates tokens in the cache that are cosine dissimilar to the current query token. This is achieved by computing the Hamming distance between binarized Gaussian projections of the current token query and cached token keys, with a projection length much smaller than the embedding dimension. We maintain a lightweight binary structure in GPU memory to facilitate these calculations. Unlike existing compression strategies that compute attention to determine token retention, LSH-E makes these decisions pre-attention, thereby reducing computational costs. Additionally, LSH-E is dynamic – at every decoding step, the key and value of the current token replace the embeddings of a token expected to produce the lowest attention score. We demonstrate that LSH-E can compress the KV cache by 30%-70% while maintaining high performance across reasoning, multiple-choice, and long-context retrieval tasks.

## 1 INTRODUCTION

The advent of large language models (LLMs) has enabled sharp improvements over innumerable downstream natural language processing (NLP) tasks, such as summarization and dialogue generation (Zhao et al., 2023; Wei et al., 2022). The hallmark feature of LLMs, the attention module (Bahdanau, 2014; Luong, 2015; Vaswani, 2017), enables contextual processing over sequences of tokens. To avoid repeated dot products over key and value embeddings of tokens, a key-value (KV) cache is maintained in VRAM to maintain these calculations. This technique is particularly popular with decoder LLMs.

However, the size of the KV cache scales quadratically with sequence length  $n$  and linearly with the number of attention layers and heads. For example, maintaining the KV cache for a sequence of 4K tokens in half-precision (FP16) can require approximately  $\sim 16$ GB of memory for most models within the Llama 3 family (Dubey et al., 2024). These memory costs are exacerbated with batched inference and result in high decoding latency (Fu, 2024). Consequently, there is significant interest in compressing the size of the KV cache to enable longer context windows and low-resource, on-device deployment.

An emerging strategy for reducing the size of the KV cache is *token eviction*. This approach drops the key and value embeddings for past tokens in the cache, skipping future attention calculations involving these tokens. Various token eviction/retention policies have been explored in recent literature, including the profiling of token type preferences (Ge et al., 2023), retention of heavy-hitter tokens (Zhang et al., 2024b;a), and dropping tokens based on the high  $L_2$  norms of their key embeddings (Devoto et al., 2024). The latter approach (Devoto et al., 2024) is intriguing as eviction decisions are performed pre-attention. However, this  $L_2$  dropout strategy only performs well on long-context retrieval tasks. It is specialized to retain only those tokens with the highest attention, which we find unsuitable for free-form reasoning tasks. Existing literature suggests that retaining

tokens with a diverse spectrum of attention scores (skewing high) is necessary (Guo et al., 2024; Zhang et al., 2024b; Long et al., 2023).

*Is there a non-attentive KV cache compression strategy that is performant over a wide variety of tasks?* This work answers this question positively by introducing a novel strategy, LSH-E, that *dynamically* determines token eviction pre-attention via locality-sensitive hashing (LSH) (Goemans & Williamson, 1995; Charikar, 2002). LSH-E evicts a past token from the cache whose key embedding is highly cosine dissimilar to the current query token embedding. The intuition behind this strategy is that high cosine dissimilarity indicates a low dot-product attention score. To efficiently scan for cosine (dis)similar tokens without performing attention, LSH-E leverages the SimHash (Charikar, 2002; Goemans & Williamson, 1995) to instead compare Hamming distances between  $c$ -length binary hashes of cached key embeddings and the current query embedding. We depict a high-level visualization of this strategy in Figure 1.

LSH-E requires minimal overhead: for a total sequence length of  $\ell$  tokens with embedding dimension  $d$ , LSH-E maintains a constant-size, low-cost binary array in GPU memory of size  $c \times k$  bytes, where  $c \ll d$  is the hash dimension and  $k \ll \ell$ . Cached tokens with key embeddings that register low Hamming similarity measurements to decoded query embeddings are gradually replaced.

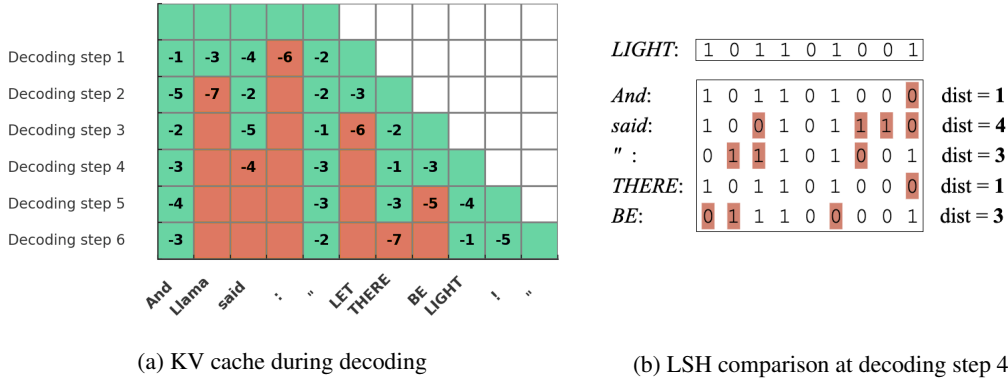


Figure 1: **An abstract visualization of LSH-E eviction strategy.** Figure 1a depicts the strategy for several decoding steps. The cache can only maintain 5 tokens due to memory constraints. At each decoding step, LSH-E projects the query embedding of the current token  $i$  and all previous key embeddings to *binary hash codes*. LSH-E then measures the negative of Hamming distances between the query *code* of token  $i$  and key *codes* of all tokens  $j$  in the cache. Each step, LSH-E evicts the key/values of the token with the lowest score (marked as red) from the cache. Figure 1b depicts the LSH comparison for decoding step 4, marking the token “said” for removal, as its high Hamming indicates low cosine similarity (and thus, low attention).

Our contributions are as follows:

- We introduce a novel *attention-free* token eviction strategy, LSH-E, that leverages locality-sensitive hashing (LSH) to quickly locate which token in the cache is the least relevant to the current query. This ranking procedure consists entirely of cheap Hamming distance calculations. The associated binary array for computing these similarities requires minimal memory overhead.
- **Novel Attention-Free Token Eviction** For a Llama 3 model, LSH-E can compress the KV cache by 30%-70% with minimal performance drop. LSH-E demonstrates high performance on reasoning tasks (GSM8K free-form Cobbe et al. (2021), MedQA free-form Cobbe et al. (2021)), long-context retrieval (Needle-in-a-Haystack, Common Word task, Ruler QA (Hsieh et al., 2024)), and multiple-choice (GSM8K MC, MedQA MC).
- **State-of-the-Art Performance** To the best of our knowledge, LSH-E achieves state-of-the-art performance for attention-free eviction across a wide variety of tasks. LSH-E outperforms  $L_2$  eviction in high-compression regimes over free response reasoning and MC tasks, while performing comparably in long-context retrieval tasks for which the  $L_2$  eviction method is designed to perform well.

- **Open-Source Implementation** Upon public release of our manuscript, we will release an open-source implementation of LSH-E through a fork of the popular cold-compress library (<https://github.com/AnswerDotAI/cold-compress>).

## 2 PRELIMINARIES

We aim to capture tokens whose query embeddings will form a large sum of dot products (i.e., attention scores) with other key embeddings, but without explicitly calculating attention. We will leverage locality-sensitive hashing (LSH) to quickly determine cosine similarities since the angle is equivalent to the dot product (for unit vectors). In this section, we review technical concepts crucial to attention and locality-sensitive hashing. We assume some base level of similarity with transformers, but we refer the reader to precise formalism (Phuong & Hutter, 2022).

**Scaled Dot-Product Attention** Consider a sequence of  $n$  tokens with  $e$ -dimensional real-valued representations  $x_1, x_2, \dots, x_n$ . Let  $Q = [q_1 q_2 \dots q_n] \in \mathbb{R}^{n \times d}$ ,  $K = [k_1 k_2 \dots k_n] \in \mathbb{R}^{d \times n}$  where  $q_i = W_q x_i$ ,  $k_i = W_k x_i$  and  $W, K \in \mathbb{R}^{d \times e}$ . The query and key projectors  $W_q$  and  $W_k$  are pre-trained weight matrices. We also define a value matrix  $V = [v_1 v_2 \dots v_n] \in \mathbb{R}^{d_{out} \times n}$  with  $v_i = W_v x_i$  with trainable  $V \in \mathbb{R}^{d_{out} \times d}$ , the scaled dot-product attention mechanism is given as

$$\text{Attention}(Q, K, V) = V \cdot \text{softmax}\left(\frac{Q^\top K}{\sqrt{d}}\right). \quad (1)$$

Typically, attention layers contain multiple heads  $\{h_i\}_{i=1}^J$  each with distinct query, key, and value projectors  $\{W_q^{(h_i)}, W_k^{(h_i)}, W_v^{(h_i)}\}_{i=1}^J$ . In a multi-head setup, attention is computed in parallel across all heads, and the outputs are concatenated together and then passed through a linear layer for processing by the next transformer block.

As  $Q, K, V$  are updated with each new incoming token, to avoid significant re-computation, the current state of  $Q^\top K$ ,  $Q$ , and  $K$  are maintained in the KV cache. Our goal is to bypass attention computation and caching for select tokens, i.e., sparsify the attention matrix  $Q^\top K$ ,  $K$ , and  $V$ .

**Locality-Sensitive Hashing** We will now describe a family of locality-sensitive hashing (LSH) functions able to efficiently approximate nearest neighbors (per cosine similarity) of key/query vectors in high-dimensional  $\mathbb{R}^d$  through comparison in a reduced  $c$ -dimensional space (per Hamming distance) with  $c \ll d$ . Here, "locality-sensitive" means points that are close together according to a distance function  $\text{dist}_d(\cdot, \cdot)$  in the ambient space remain close per another distance function  $\text{dist}_c(\cdot, \cdot)$  in the lower-dimensional space with high-probability. For a rigorous treatment of LSH functions, see (Andoni et al., 2018; Charikar, 2002).

Formally for our setup,  $\text{dist}_d(x, y) \triangleq \cos \theta_{x,y} = \frac{x^\top y}{\|x\| \|y\|}$  and  $\text{dist}_c(p, q) \triangleq d_H(p, q)$  which denotes the Hamming distance. We will project each vector from  $\mathbb{R}^d$  into  $\mathbb{Z}_2^c$ , the space of  $c$ -bit binary strings (which is often referred to as a *binary hash code*). To acquire a  $c$ -bit long hash code from an input vector  $x \in \mathbb{R}^d$ , we define a random projection matrix  $R \in \mathbb{R}^{c \times d}$  whose entries are independently sampled from the standard normal distribution  $\mathcal{N}(0, 1)$ . We then define

$$h(x) = \text{sgn}(Rx), \quad (2)$$

where  $\text{sgn}(\cdot)$  (as an abuse of conventional notation) is the element-wise Heaviside step function:

$$\text{sgn}(x) := \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}.$$

For two unit vectors  $x, y \in \mathbb{R}^d$  we have that,

$$\frac{1}{c} \cdot \mathbb{E}[d_H(h(x), h(y))] = \frac{\theta_{x,y}}{\pi}, \quad (3)$$

where  $\theta_{x,y} = \arccos(\cos(\theta_{x,y}))$ . We do not prove equation 3 in this work; see Theorem §3.1 in (Goemans & Williamson, 1995, Theorem 3.1). In particular, if  $x$  and  $y$  are close in angle, the

Hamming distance between  $h(x)$  and  $h(y)$  is low in expectation. Increasing the hash dimension  $c$  reduces variance.

The geometric intuition behind this LSH scheme is the following: each row  $R_{:,i}$  of  $R$  defines a random hyperplane in  $\mathbb{R}^d$ . The Heaviside function  $\text{sgn}(\cdot)$  indicates whether  $x$  is positively or negatively oriented with respect to the hyperplane  $R_{:,i}$ . Thus, the  $c$  hyperplanes divide the  $d$  dimensional space into multiple partitions, and the resulting  $c$ -dimensional hash code is an index into one of the partitions in which  $x$  is located. Therefore, vectors with the same or similar hash codes lie in the same or close-by partitions and, therefore, are likely similar in angle.

**Remark** LSH is conventionally used to find the set of approximate nearest neighbors of an input  $x \in \mathbb{R}^d$  against a large collection of candidates  $Y = \{y_i\}_{i=1}^N$  (Andoni et al., 2018). In particular, the user searches for  $\arg \min_i d_H(h(x), h(y_i))$  – the closest matching code. As we will see in Section 3, we are instead interested in  $\arg \max_i d_H(h(x), h(y_i))$ : the token with the most dissimilar hash code to the query.

## 2.1 RELATED WORKS

**KV Cache Compression** Many popular compression strategies adopt an *eviction* approach, which removes embeddings from the KV cache. H<sub>2</sub>O (Zhang et al., 2024b) and Scissorhands (Liu et al., 2024b) calculate token importance by their accumulated attention scores and keep the ‘heavy hitters’ in the cache. FastGen (Ge et al., 2023) performs a profiling pass before the generation stage that assigns to each head, according to the head’s attention patterns, a pruning policy which only retains categories of tokens (punctuation, special, etc.) favored by the head. These eviction strategies depend on the computation of attention scores for their policy. An attention-free  $L_2$  dropout method (Devoto et al., 2024), which we compare ourselves to in this work, uses the observation that high-attention tokens tend to have low  $L_2$  key norms to approximately keep important tokens in cache.

Other methods seek to merge KV caches across heads, such as grouped query attention (GQA) (Ainslie et al., 2023; Dubey et al., 2024). KVMerger (Wang et al., 2024) and MiniCache (Liu et al., 2024a), which searches for similarity between tokens in consecutive attention layers and subsequently merges KV cache entries across these layers. While these consolidation approaches prevent memory complexity associated with KV caches from scaling with depth or multi-head attention, the size of any singular cache still tends to scale with sequence length.

**Memory Efficient Transformers** Multi-Query Attention (Shazeer, 2019) and Grouped Query Attention (Ainslie et al., 2023) reduce the number of key-value matrices by sharing them across multiple query heads to save KV cache memory usage. However, they require re-training or up-training the LLM. Cache quantization methods (Hooper et al., 2024; Sheng et al., 2023) reduce the KV cache size by compressing the hidden dimension instead of along the sequence dimension but can result in information loss. Linear Transformer (Katharopoulos et al., 2020) reduces memory usage by replacing the softmax attention with linear kernels and, therefore, achieves constant memory requirement. Similar to our work, Reformer (Kitaev et al., 2020) employs LSH to find similar tokens as a way to replace the softmax attention. It creates hash buckets of tokens that form local attention groups and only attends to tokens in the same and neighboring buckets. However, this makes Reformer vulnerable to missing important tokens due to hash collision or boundary issues, and therefore, it must use multiple hash tables to mitigate this issue.

## 3 LSH-E: A LOCALITY-SENSITIVE EVICTION STRATEGY

We now formalize our eviction method reflected in Algorithm 1. We assume that the KV cache has a limited and fixed budget and conceptually divide the KV cache management during LLM inference into two stages: the initial Prompt Encoding Stage and then a Decoding Stage (i.e., generation).

Let  $C$  be a constant and fixed cache budget,  $\mathcal{K}$  be the key cache, and  $\mathcal{V}$  be the V cache in a K-V attention head. We define our eviction policy as a function

$$\mathcal{K}_t, \mathcal{V}_t, \mathcal{H}_t \leftarrow P(q, \mathcal{K}_{t-1}, \mathcal{V}_{t-1}, \mathcal{H}_{t-1}) \quad (4)$$

where  $\mathcal{H}_t \in \{0, 1\}^{b \times C}$  is a hash table that contains hash codes of keys in  $\mathcal{K}$ . We then define a function  $F_{score}$  to assign a score for each key inside the K cache.  $F_{score}$  outputs an array which

contains the negative of hamming distances  $d_H$  between the hash code of a query vector  $q$  and columns of  $\mathcal{H}$ , which are hash codes of all non-evicted keys.

$$F_{score}(q, \mathcal{K}) = -d_H(h(q), \mathcal{H}) \quad (5)$$

The eviction index  $e_t$  at any step  $t$  is selected as the index with the lowest score:

$$e_t \leftarrow \arg \min F_{score}(q_{t-1}, \mathcal{H}_{t-1}) \quad (6)$$

which points to the key that is most distant from the query vector at time step  $t$ . Entries at index  $e_t$  from the  $\mathcal{K}$  and  $\mathcal{V}$  are evicted and  $\mathcal{H}$  is updated (step 3-6 of Algorithm 1).

---

**Algorithm 1** LSH-E (timestep  $t$ )

---

**Require:** query  $q$ , key  $k$ , value  $v$ , key cache  $\mathcal{K}$ , value cache  $\mathcal{V}$ , hash table  $\mathcal{H}$

- 1:  $e_t \leftarrow \arg \min F_{score}(q_t, \mathcal{H}_{t-1})$  ▷ Determine eviction index  $e_t$
  - 2: **del**  $\mathcal{K}_{t-1}^{e_t}, \mathcal{V}_{t-1}^{e_t}, \mathcal{H}_{t-1}^{e_t}$  ▷ Remove entries at index  $e_t$  from KV cache and hash table
  - 3:  $\mathcal{K}_t \leftarrow \mathcal{K}_{t-1} \cup k_t$  ▷ Update key cache
  - 4:  $\mathcal{V}_t \leftarrow \mathcal{V}_{t-1} \cup v_t$  ▷ Update value cache
  - 5:  $\mathcal{H}_t \leftarrow \mathcal{H}_{t-1} \cup h(k_t)$  ▷ Add hash of  $k_t$  to the hash table
  - 6:  $A \leftarrow \text{Attention}(q, \mathcal{K}_t, \mathcal{V}_t)$  ▷ Calculate attention
- 

**Prompt Encoding Stage** During the prompt encoding stage, the model processes the prompt,  $x_{prompt} = [x_1, \dots, x_N] \in \mathbb{R}^{N \times d}$ . The KV cache and the hash table are first filled to full by the first  $C$  tokens.  $\mathcal{K}_0 = \{k_1, \dots, k_C\}$ ,  $\mathcal{V}_0 = \{v_1, \dots, v_C\}$ ,  $\mathcal{H}_0 = h(\mathcal{K}_0) = \bigcup_{i \in [1, C]} h(k_i)$ . We then set  $t \leftarrow C + 1$ , and begin Algorithm 1.

**Decoding Stage** Let  $x_{decoding} = [z_1, \dots, z_T] \in \mathbb{R}^{T \times d}$  be the generated tokens during auto-regressive decoding. In the decoding stage, we continue Algorithm 1 by setting  $t < -N + 1$ . The generation completes at time step  $N + T$ .

**Complexity** Our strategy assumes a fixed memory budget, and therefore, uses constant memory. The computation overhead per time step is also constant, because  $F_{score}$  is calculated for a constant  $C$  number of key vectors in the cache. The extra memory overhead that LSH-E introduces to each attention head is the hash table  $\mathcal{H}$ , which only uses  $C * b$  bits of space and is independent of the sequence length. The hash table is stored on GPU memory and does not introduce any latency bottlenecks associated with CPU-to-GPU streaming (Strati et al., 2024).

## 4 EXPERIMENTS

**Tasks** We evaluated our LSH eviction strategy across various tasks to demonstrate its effectiveness in reducing the memory cost of the KV cache while preserving the language quality of the generated text. Our experiments are split into three main categories: free response question answering, multiple choice, and long-context retrieval. Our long context retrieval tasks include the multi-key needle-in-a-haystack task and the common words task from (Hsieh et al., 2024). Question answering tasks include GSM8K (Cobbe et al., 2021) and MedQA (Jin et al., 2021).

**Metrics** The question-answering tasks were evaluated using BERTScore (which includes precision, recall, and F1 scores), ROUGE (ROUGE-1, ROUGE-2 and ROUGE-L and ROUGE-Lsum), and GPT4-Judge. GPT-4 was prompted to look at both the model prediction and the ground truth answer, then provide a score from 1 - 5 on the coherence, faithfulness, and helpfulness of the answer in addition to similarity between the prediction and ground truth (we named this metric GPT4-Rouge). In this section, we report the average of these four scores. For details on individual scores, please see Appendix A. For the system prompts given to GPT-4, refer to Appendix B.2. For multiple-choice tasks, we use accuracy as our metric. The metric used to evaluate long context retrieval tasks is the string matching score from Hsieh et al. (2024), whose definition is in Appendix B.1.

**Configuration and Setup** We conducted experiments using Meta’s Llama3 8B-Instruct model (Dubey et al., 2024). Our method is agnostic to grouped-query attention, so we used the default group size of 4. The maximum sequence length was set to the sum of the maximum prompt length and the maximum number of allowed generated tokens needed for each task. We conducted experiments using cache budgets of 10%, 30%, 50%, 70%, and 90% of the full KV cache. Based on insights from (Xiao et al., 2023; Child et al., 2019; Beltagy et al., 2020), we also keep the most recent 10 tokens and the first 4 tokens of the prompt always in the KV cache. We chose the  $L_2$  norm-based eviction method (Devoto et al., 2024) as a baseline for comparison because it is also an eviction method that does not depend on the attention score. All experiments were conducted on the Google Cloud Platform G2 instances with Nvidia L4 24GB graphics cards.

#### 4.1 FREE RESPONSE QUESTION ANSWERING

We tested our strategy against tasks that require generating accurate answers using multi-step reasoning. Specifically, we used the GSM8K and MedQA datasets to assess language quality for each strategy, given a constrained KV cache budget. Both tasks are used to test the potential side effects of compression on the LLM’s reasoning ability.

##### 4.1.1 GSM8K FREE RESPONSE RESULTS

GSM8K consists of grade-school-level math problems that typically require multiple reasoning steps. As shown in Figure 2, our LSH eviction strategy consistently outperforms the  $L_2$  norm-based method across various cache sizes. Notably, even when the KV cache budget is set to 50% of the full capacity, the LSH eviction strategy maintains a high answer quality, with minimal degradation in BERTScore F1, ROUGE-L, and GPT4-Judge scores.

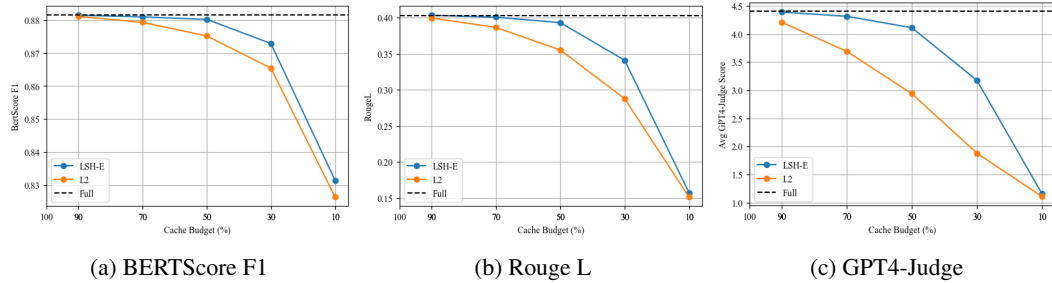


Figure 2: **GSM8K Question Answering Performance.** We measure BERTScore F1, Rouge-L, and GPT4-Judge for different cache budgets on a grade school math task. LSH-E outperforms  $L_2$  for all three metrics for every budget, with sharp differences for the 50% and 30% compression.

##### 4.1.2 MEDQA FREE RESPONSE RESULTS

MedQA is a free response multiple choice question answering dataset collected from professional medical board exams. We sample 100 questions from this dataset. Each question has 5 choices and only one correct answer, along with ground truth explanations and reasoning steps. Figure 3 illustrates that LSH-E performs better than  $L_2$  eviction for all budgets tested. For both datasets, LSH-E produced more coherent and helpful answers across all cache budgets than  $L_2$  eviction per Table 7. For detailed experiment results, please refer to Appendix A.

#### 4.2 MULTIPLE CHOICE QUESTION ANSWERING

We evaluated our method on multiple-choice versions of GSM8K and MedQA. Multiple choice is a more difficult test of a model’s reasoning capability under the constraint of cache compression, as it takes away the ability to use intermediate results in the generated text. The model has to keep useful tokens during prompt compression in order to pick the correct answer choice.

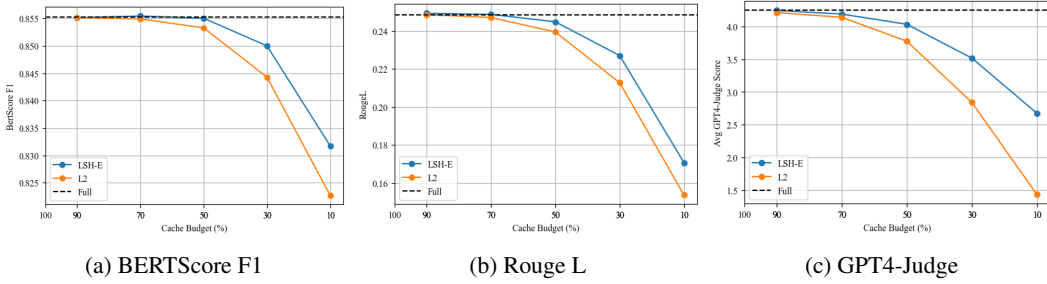


Figure 3: **MedQA Question Answering Performance.** We measure BertScore F1, Rouge-L, and GPT4-Judge for different cache budgets on a medical exam task. LSH outperforms  $L_2$  for all three metrics for every budget, with a significantly higher performance for the 30% and 10% budgets.

#### 4.2.1 GSM8K MULTIPLE CHOICE RESULTS

For the multiple choice experiments, LSH significantly outperforms  $L_2$  for cache budgets of 30% and 50%. As shown in Figure 4a, the  $L_2$  method’s accuracy drops significantly at smaller cache sizes, while the performance of LSH-E does not significantly drop until the cache budget is set at 10%.

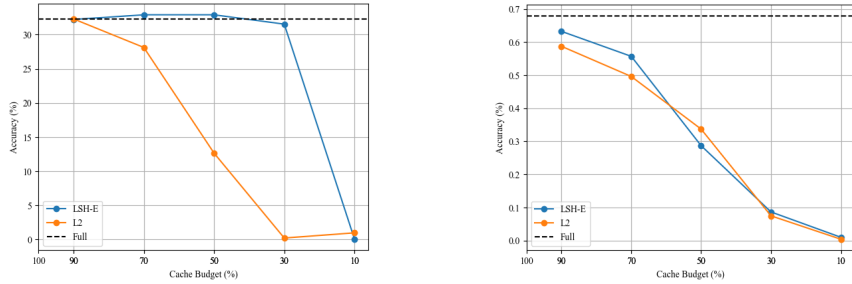


Figure 4: **Multiple Choice Tasks Performance.** On GSM8K, LSH-E outperforms the baseline full cache on GSM8K at 70% and 50% cache budgets and significantly outperforms  $L_2$  at 70%, 50%, and 30%. LSH-E performs on par with  $L_2$  overall on MedQA with higher performance at 90% (near uncompressed performance) and 70% budget and slightly lower performance at 50% budget.

#### 4.2.2 MEDQA MULTIPLE CHOICE RESULTS

Per Figure 4b, the MedQA multiple choice experiment, LSH offers better performance than  $L_2$  eviction for all tested cache budgets except for 50%. Performance between both methods is highly similar at lower budgets.

### 4.3 LONG-CONTEXT RETRIEVAL

To evaluate LSH-E’s ability to retain and retrieve important pieces of information from long contexts, we used the Needle-in-a-Haystack and Common Words tasks from Hsieh et al. (2024). These tests benchmark the ability of a compression strategy to retain important tokens inside the KV cache within a large, complex stream of context. The  $L_2$  eviction method (Devoto et al., 2024) is specifically designed for these types of benchmarks, so closely matching its performance will demonstrate the task versatility of LSH-E.

#### 4.3.1 NEEDLE-IN-A-HAYSTACK

In this task (evaluated with a 4k context length), the model must extract specific information buried within a large body of text. As illustrated in Figure 5b, LSH-E slightly outperforms  $L_2$  at every

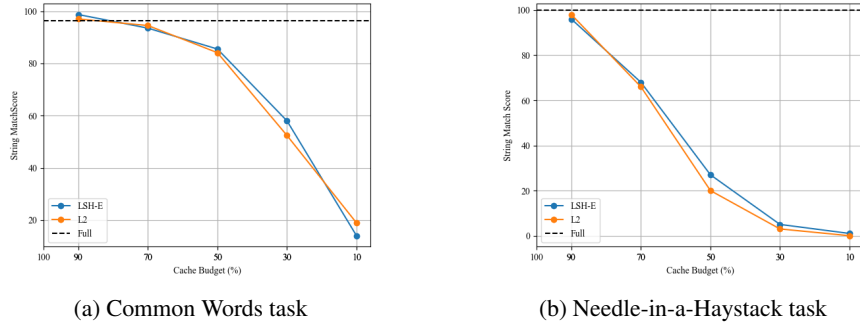


Figure 5: **Long-Context Tasks.** We measure string-matching scores for two long-context retrieval tasks. LSH-E performs on par with  $L_2$  on the Common Words task with slightly higher performance at a 30% cache budget and slightly lower performance at a 10% budget. For the Needle-in-a-Haystack task, LSH-E performs on par with  $L_2$  with slightly higher performance at a 50% cache budget.

cache budget except for 90%, and both methods see a sharp drop in the ability to recall the “needle” (a small, targeted piece of context) after the cache budget drops to 50% and lower. LSH-E outperforms  $L_2$  for these smaller cache sizes.

#### 4.3.2 COMMON WORDS

In the Common Words task, the model must identify the most frequent words from a long list. Figure 5a demonstrates that LSH-E performs on par with  $L_2$  eviction in general and slightly better at 30%, 50%, and 90% cache budget. Both methods outperform the full cache model at 90% cache size, indicating that some cache compression can actually increase performance. Neither method experienced a significant drop in performance until the cache budget was reduced to 30%.

#### 4.4 MEMORY USAGE

Table 2 compares the memory usage of the KV cache and relevant data structures of  $L_2$  and LSH-E on the GSM8K and MedQA question answering experiments. LSH-E maintains  $\mathcal{H}$ , a binary hash matrix of the attention keys in memory and, therefore, has slightly higher memory usage than  $L_2$  eviction. Our implementation uses 8 bits for binary values instead of 1 bit. Using 1-bit binary numbers would reduce the memory overhead of LSH-E by a factor of 8 and narrow the difference in memory usage between LSH-E and  $L_2$ .

Table 1: **LSH Hash Dimension Ablation.** We assesses GSM8K Question Answering performance for different LSH dimensions. The cache budget is fixed at 50%. LSH dimension does significantly impact performance. Small LSH dimensions slightly outperform larger LSH dimensions.

LSH Dim	BERTScore F1	Rouge L	GPT4 Judge	Compression Ratio	Cache Memory (GB)
4	0.8807	0.3974	4.3833	0.3728	2.8062
8	0.8802	<b>0.3975</b>	<b>4.4113</b>	0.3734	2.8355
16	<b>0.8807</b>	0.3972	4.3753	0.3716	2.8941
24	0.8802	0.3951	4.3733	0.3711	2.9527
32	0.8796	0.3926	4.3220	0.3710	3.0113
64	0.8797	0.3900	4.2333	0.3702	3.2456



Table 2: **GSM8K and MedQA Question Answering KV Cache Memory Usage.** LSH-E maintains a binary hash matrix of attention keys in memory and, therefore, has slightly higher memory usage than  $L_2$ . Our implementation uses 8-bits for binary values instead of 1-bit. Using 1-bit binary numbers will reduce the memory overhead of LSH-E by a factor of 8 and decrease the difference in memory usage between LSH-E and  $L_2$ .

Cache Budget (%)	Strategy	GSM8K		MedQA	
		Compression Ratio	Cache Memory (GB)	Compression Ratio	Cache Memory (GB)
10	$L_2$	0.8355	0.7603	0.9289	2.5342
	LSH-E	0.8380	0.8120	0.8812	2.6338
30	$L_2$	0.6234	1.7740	0.6957	7.3492
	LSH-E	0.6018	1.8531	0.6360	7.5786
50	$L_2$	0.3968	2.7876	0.4175	12.1641
	LSH-E	0.3716	2.8941	0.3901	12.5235
70	$L_2$	0.1967	3.8013	0.1803	17.2325
	LSH-E	0.1857	3.9351	0.1740	17.7285
90	$L_2$	0.0859	4.8150	0.0498	22.0474
	LSH-E	0.0823	4.9761	0.0483	22.6734
100	Full	0.0000	12.6934	0.0000	51.1181

#### 4.5 ABLATION ON LSH DIMENSION

To determine the effect of the LSH compression dimension, we conducted an ablation study using the GSM8K free response dataset. Fixing the cache budget to 50%, we tested LSH dimensions of 4, 8, 16, 32 and 64 bits. Table 1 shows the results. The choice of LSH dimension does not significantly impact performance. In fact, 8 bits performed the best, but not noticeably better than higher dimensions. This demonstrates that LSH-E does not require a high hashing dimension and can be executed with minimal storage overhead. When using 8 bits, the storage overhead is 1 byte  $\times$  cache size. For example, in a Llama3 70B-Instruct deployment with 80 layers, 8 KV-heads, sequence length of 8192, batch size of 8 and 50% cache budget, LSH dimension of 8-bits, we have that 16-bits and 32-bits only use an extra 20MB, 40MB, and 80MB respectively, which are significantly smaller than the KV cache size of 640GB.

#### 4.6 ATTENTION LOSS RATIO ANALYSIS

We perform an attention loss ratio (ALR) analysis between LSH-based ranking and  $L_2$ -based ranking. Our implementation is an adaptation of the methodology described in Devoto et al. (2024). This section explores how much of the uncompressed attention matrix is preserved between LSH-E and the  $L_2$  eviction strategy in Devoto et al. (2024).

Compressing the KV cache entails dropping KV pairs. Per (Devoto et al., 2024), we can define the attention loss caused by the compression as the sum of the attention scores associated with the dropped KV pairs in layer  $l$  and head  $h$  via the equation  $L_{l,h}^m = \sum_{p \in D_{l,h}^m} a_{l,h,p}$ , where  $a_{l,h,p}$  is the average attention score at position  $p$  for layer  $l$  and head  $h$ , and  $D_{l,h}^m$  denotes the positions of the  $m$  dropped KV pairs, with  $|D_{l,h}^m| = m$ . We process a selection of prompts and examine how proposed evictions by the  $L_2$  eviction strategy and LSH-E would affect the sum of attention scores.

To quantify the additional attention loss introduced by using an alternative ranking method (such as  $L_2$  norm or LSH-E’s  $F_{score}$ ) instead of the true attention-based ranking, we define the cumulative attention loss difference as:

$$Y_{l,h} = \sum_{m=1}^n (L_{l,h}^m - L_{l,h,\text{ref}}^m), \quad (7)$$

where  $L_{l,h,\text{ref}}^m$  is the cumulative attention loss when dropping the KV pairs with the actual lowest attention scores. The value  $Y_{l,h}$  is non-negative, and a lower value indicates that the ranking method closely approximates non-compressed attention. Figure 6 depicts the ALR for the  $L_2$  eviction rankings and an LSH ranking.

It is important to note that LSH-E is not designed to produce a global ranking among as the  $L_2$  method is designed to do (via a low-to-high ordering of all  $L_2$  key norms). LSH-E ranks the importance of past tokens with regards to the current token – and this ranking changes every step. To simulate a comparison, we record the average Hamming distance between the key code of token  $i$  and the query codes of all tokens  $j > i$ . We then sort tokens from lowest to highest average Hamming distance. Figure 6a reflects the ALR according to this ranking system. The  $L_2$  ranking exclusively prefers high-attention tokens, while the LSH ranking prefers medium-to-high-attention tokens. Based on our empirical results in Section 4, the selection of tokens over a spectrum of attention scores skewing towards high results in greater task versatility compared to the  $L_2$  eviction.

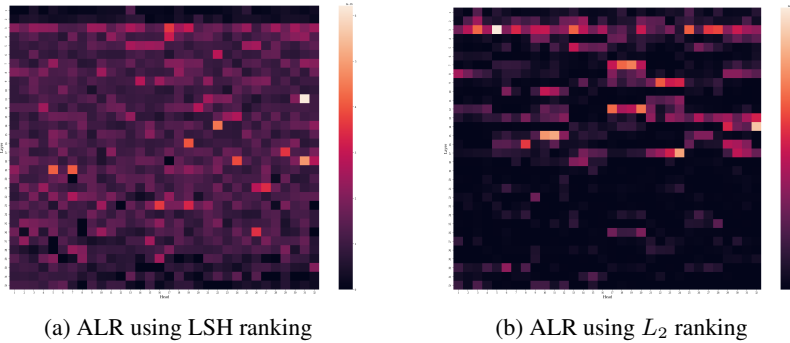


Figure 6: **Attention Loss Ratio (ALR)**. We compare how the eviction strategy of LSH-E and the  $L_2$  method (Devoto et al., 2024) affects the ALR per equation 7. Our tested model is Llama3-8B-Instruct, which contains 32 heads and 32 attention layers. Cell  $(i, j)$  depicts the ALR of head  $i$  in attention layer  $j$ . A darker score indicates a lower ALR. The  $L_2$  method exhibits extremely low ALR, thus indicating exclusive preference for high-attention tokens. LSH-E prefers to select medium-to-high attention tokens.

## 5 DISCUSSION & CONCLUSION

In this paper, we introduce LSH-E, a novel attention-free eviction strategy for KV cache compression in transformer-based LLMs. By leveraging locality-sensitive hashing (LSH) to approximate cosine similarity, LSH-E dynamically determines which tokens to evict from the cache without performing costly attention calculations. Our experiments demonstrate that LSH-E can achieve 30-70% compression of the KV cache while maintaining strong performance across various tasks, including free-response Q&A, multiple-choice Q&A, and long-context retrieval.

The key advantage of LSH-E lies in its ability to efficiently compress the KV cache pre-attention, enabling significant memory savings and faster inference times. Compared to traditional strategies like  $L_2$  norm-based eviction (Devoto et al., 2024), LSH-E excels particularly in reasoning and multiple-choice tasks, where maintaining a diverse set of tokens in the cache is crucial for generating accurate and coherent responses.

There are several potential areas for future work. Investigating hybrid approaches that combine LSH-based eviction with attention-based mechanisms such as (Zhang et al., 2024b; Ge et al., 2023) could offer a middle ground between computational efficiency and retention of high-importance tokens. Further, reducing the overhead associated with maintaining binary hash codes (e.g., by optimizing bit precision) could further enhance the applicability of LSH-E to memory-constrained environments.

## REFERENCES

- 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.
- Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pp. 3287–3318. World Scientific, 2018.
- Dzmitry Bahdanau. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pp. 380–388, 2002.
- Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. A simple and effective  $l_2$  norm-based strategy for kv cache compression. *arXiv preprint arXiv:2406.11430*, 2024.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- Yao Fu. Challenges in deploying long-context transformers: A theoretical peak performance analysis. *arXiv preprint arXiv:2405.08944*, 2024.
- Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.
- Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6): 1115–1145, 1995.
- Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. Attention score is not all you need for token importance indicator in kv cache reduction: Value also matters. *arXiv preprint arXiv:2406.12335*, 2024.
- Ankit Gupta, Guy Dar, Shaya Goodman, David Ciprut, and Jonathan Berant. Memory-efficient transformers via top- $k$  attention. *arXiv preprint arXiv:2106.06899*, 2021.
- Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*, 2024.
- Cheng-Ping Hsieh, Simeng Sun, Samuel Krman, Shantanu Acharya, Dima Rekesch, Fei Jia, and Boris Ginsburg. Ruler: What’s the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.
- Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

- Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.
- Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.
- Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Mini-cache: Kv cache compression in depth dimension for large language models. *arXiv preprint arXiv:2405.14366*, 2024a.
- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhao 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, 2024b.
- Sifan Long, Zhen Zhao, Jimin Pi, Shengsheng Wang, and Jingdong Wang. Beyond attentive tokens: Incorporating token importance and diversity for efficient vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10334–10343, 2023.
- Minh-Thang Luong. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.
- Mary Phuong and Marcus Hutter. Formal algorithms for transformers. *arXiv preprint arXiv:2207.09238*, 2022.
- Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.
- Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.
- Foteini Strati, Sara McAllister, Amar Phanishayee, Jakub Tarnawski, and Ana Klimovic. D\`ej\`avu: Kv-cache streaming for fast, fault-tolerant generative llm serving. *arXiv preprint arXiv:2403.01876*, 2024.
- A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
- Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*, 2024.
- Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.
- Zhenyu Zhang, Shiwei Liu, Runjin Chen, Bhavya Kailkhura, Beidi Chen, and Atlas Wang. Q-hitter: A better token oracle for efficient llm inference via sparse-quantized kv cache. *Proceedings of Machine Learning and Systems*, 6:381–394, 2024a.
- 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, 2024b.
- Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

## APPENDIX

## A QUESTION ANSWERING GRANULAR EXPERIMENT RESULTS

Table 3: GSM8K and MedQA Question Answering BERTScore

Cache Budget (%)	Strategy	GSM8K			MedQA		
		Precision	Recall	F1	Precision	Recall	F1
10	$L_2$	0.8585	0.7983	0.8270	0.8330	0.8126	0.8226
	LSH-E	<b>0.8602</b>	<b>0.8067</b>	<b>0.8323</b>	<b>0.8570</b>	<b>0.8080</b>	<b>0.8317</b>
30	$L_2$	0.8853	0.8487	0.8665	0.8554	0.8336	0.8443
	LSH-E	<b>0.8934</b>	<b>0.8557</b>	<b>0.8740</b>	<b>0.8665</b>	<b>0.8343</b>	<b>0.8500</b>
50	$L_2$	0.8907	0.8611	0.8756	0.8659	0.8412	0.8533
	LSH-E	<b>0.8970</b>	<b>0.8652</b>	<b>0.8807</b>	<b>0.8689</b>	<b>0.8417</b>	<b>0.8551</b>
70	$L_2$	0.8946	0.8653	0.8796	0.8679	0.8425	0.8549
	LSH-E	<b>0.8964</b>	<b>0.8666</b>	<b>0.8812</b>	<b>0.8687</b>	<b>0.8427</b>	<b>0.8555</b>
90	$L_2$	0.8961	0.8665	0.8810	0.8681	0.8427	0.8552
	LSH-E	<b>0.8965</b>	<b>0.8670</b>	<b>0.8814</b>	<b>0.8682</b>	0.8427	0.8552
100	Full	0.8967	0.8672	0.8816	0.8682	0.8428	0.8553

Table 4: GSM8K Question Answering Rouge

Cache Budget (%)	Strategy	Rouge 1	Rouge 2	Rouge L	Rouge Lsum
10	L2	0.1961	0.0494	0.1533	0.1795
	LSH-E	<b>0.2044</b>	<b>0.0510</b>	<b>0.1558</b>	<b>0.1840</b>
30	L2	0.3979	0.1515	0.2924	0.3410
	LSH-E	<b>0.4529</b>	<b>0.1900</b>	<b>0.3471</b>	<b>0.3882</b>
50	L2	0.4800	0.2070	0.3588	0.4109
	LSH-E	<b>0.5133</b>	<b>0.2379</b>	<b>0.3972</b>	<b>0.4404</b>
70	L2	0.5103	0.2337	0.3907	0.4364
	LSH-E	<b>0.5213</b>	<b>0.2424</b>	<b>0.4040</b>	<b>0.4460</b>
90	L2	0.5191	0.2403	0.4014	0.4438
	LSH-E	<b>0.5224</b>	<b>0.2433</b>	<b>0.4055</b>	<b>0.4465</b>
100	Full	0.5239	0.2449	0.4054	0.4474

## B METRICS AND PROMPTS

## B.1 STRING MATCH SCORE

The string matching score is calculated as:

$$\text{String Matching Score} = \frac{\text{Number of correctly matched characters in predicted string}}{\text{Total number of characters in GT}} \times 100$$

Table 5: GSM8K Question Answering GPT4-Judge

Cache Budget (%)	Strategy	Similarity to GT	Coherence	Faithfulness	Helpfulness
10	L2 LSH-E	1.0020 <b>1.0360</b>	1.3140 <b>1.4860</b>	1.0940 <b>1.2000</b>	1.0320 <b>1.1100</b>
30	L2 LSH-E	1.4300 <b>2.6920</b>	2.5340 <b>3.8880</b>	1.9700 <b>3.3900</b>	1.9820 <b>3.3680</b>
50	L2 LSH-E	2.3060 <b>3.5660</b>	3.5760 <b>4.5780</b>	3.1420 <b>4.2880</b>	3.1120 <b>4.3040</b>
70	L2 LSH-E	3.1200 <b>3.8400</b>	4.2660 <b>4.6960</b>	3.9520 <b>4.4540</b>	3.9420 <b>4.4820</b>
90	L2 LSH-E	3.6060 <b>3.9120</b>	4.5660 <b>4.7240</b>	4.3140 <b>4.5200</b>	4.3560 <b>4.5420</b>
100	Full	3.9240	4.7340	4.5760	4.5980

Table 6: MedQA Question Answering Rouge

Cache Budget (%)	Strategy	Rouge 1	Rouge 2	Rouge L	Rouge Lsum
10	L2 LSH-E	0.3043 <b>0.3457</b>	0.0717 <b>0.1102</b>	0.1536 <b>0.1706</b>	0.2885 <b>0.3242</b>
30	L2 LSH-E	0.4285 <b>0.4495</b>	0.1461 <b>0.1701</b>	0.2128 <b>0.2271</b>	0.4070 <b>0.4256</b>
50	L2 LSH-E	0.4736 <b>0.4808</b>	0.1845 <b>0.1935</b>	0.2395 <b>0.2449</b>	0.4495 <b>0.4554</b>
70	L2 LSH-E	0.4837 <b>0.4871</b>	0.1943 <b>0.1974</b>	0.2472 <b>0.2488</b>	0.4580 <b>0.4611</b>
90	L2 LSH-E	0.4866 <b>0.4870</b>	0.1966 <b>0.1973</b>	0.2487 <b>0.2494</b>	0.4606 <b>0.4610</b>
100	Full	0.4865	0.1976	0.2484	0.4602

Table 7: MedQA Question Answering GPT4-Judge

Cache Budget (%)	Strategy	Similarity to GT	Coherence	Faithfulness	Helpfulness
10	L2 LSH-E	1.1031 <b>1.9695</b>	1.6955 <b>3.5167</b>	1.6395 <b>2.6650</b>	1.2829 <b>2.5472</b>
30	L2 LSH-E	1.9391 <b>2.5108</b>	3.6326 <b>4.4145</b>	2.9420 <b>3.5334</b>	2.8428 <b>3.6130</b>
50	L2 LSH-E	2.8497 <b>3.0216</b>	4.5108 <b>4.7299</b>	3.7967 <b>4.1385</b>	3.9499 <b>4.2544</b>
70	L2 LSH-E	3.1945 <b>3.2318</b>	4.7554 <b>4.8094</b>	4.2348 <b>4.2917</b>	4.3851 <b>4.4342</b>
90	L2 LSH-E	3.2652 <b>3.2908</b>	4.8183 <b>4.8389</b>	4.3183 <b>4.3546</b>	4.4578 <b>4.5069</b>
100	Full	3.3369	4.8173	4.3418	4.5000

## B.2 GPT-4-JUDGE PROMPT

For the GPT-4-Judge metric used in evaluating free response question answering tasks, we accessed the GPT-4o model through OpenAI’s API.

For the GPT4-Rouge metric, the prompt given to the model is:

```
You are shown ground-truth answer(s) and asked to judge the quality of an
LLM-generated answer.
Assign it a score from 1-5 where 1 is the worst and 5 is the best based
on how similar it is to the ground truth(s).
Do NOT explain your choice. Simply return a number from 1-5.
```

```
====GROUND TRUTHS====
{labels}
```

```
====ANSWER====
{prediction}
```

For the other three GPT4-Judge based on criteria, the prompt given to the model is:

```
You are shown a prompt and asked to assess the quality of an LLM-
generated answer on the following dimensions:
```

```
===CRITERIA===
{criteria}
```

```
Respond with "criteria: score" for each criterion with a newline for each
criterion.
Assign a score from 1-5 where 1 is the worst and 5 is the best based on
how well the answer meets the criteria.
```

```
====PROMPT====
{prompt}
```

```
====ANSWER====
{prediction}
```

The list of criteria is:

```
CRITERIA = {
  "helpful": "The answer executes the action requested by the prompt
without extraneous detail.",
  "coherent": "The answer is logically structured and coherent (ignore
the prompt).",
  "faithful": "The answer is faithful to the prompt and does not contain
false information.",
}
```

## C ATTENTION SCORES AND KEY NORMS VISUALIZATION

We further examine the method of our chief competitor, the  $L_2$  eviction method (Devoto et al., 2024). In particular, in Figure 7 we examine the key-norm-attention correlation suggested by the authors. Indeed, low key-norms, even across prompts, demonstrate a strong correlation with attention score.

## D ANALYSIS OF THE RELATIONSHIP BETWEEN ATTENTION SCORES AND LSH HAMMING DISTANCE

In this section, we follow up on our ALR in Section 4.6. We analyze the relationship between attention scores and average LSH Hamming distances using 50 randomly selected prompts from GSM8K. We stress that this metric does not perfectly capture the “ranking” system of LSH-E (which cannot perform a global/full-sequence token-importance ranking like  $L_2$  eviction).

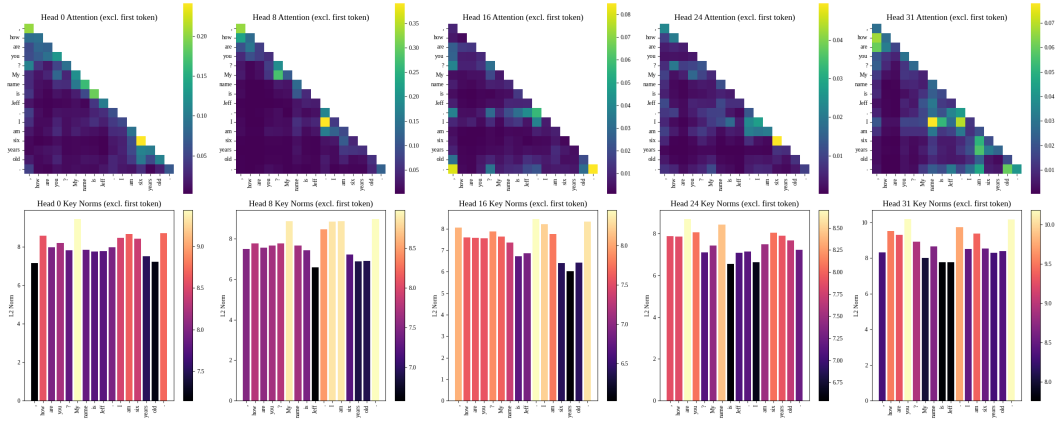


Figure 7: **Attention and Key Norms.** Attention scores and corresponding  $L_2$  norms of key vectors (excluding the first token) for a sample of heads (0,8,16,24,31) in the 8th layer for a sample input sequence. Each subplot shows the attention heatmap (top) and the corresponding key norm values (bottom) for a particular head, allowing for a direct comparison between attention patterns and key norm values across different heads.

For each prompt, we performed the following:

1. **Captured States:** Extracted normalized key and query vectors from every layer and head combination after applying rotary positional embeddings.
2. **Applied Random Projections:** Applied multiple random Gaussian projections, varying the projection length (number of bits). We tested with projection lengths of 8, 16, 24, and 32.
3. **Computed Hamming Distances:** Computed the Hamming distances between the projected and binarized vectors and averaged this over multiple projections to mitigate the randomness that LSH introduces and to obtain a more stable estimate of the Hamming distances.
4. **Computed Correlations:** Calculated the Pearson correlation coefficient between the attention scores and the inverted average Hamming distance for each layer and head combination and for each projection length.

## D.1 RESULTS

The average Pearson correlation between the attention scores and the inverted average Hamming distances is  $0.2978 \pm 0.1947$ . Table 8 and Figure 8a detail the average Pearson correlation per projection length.

Table 8: Average Pearson correlation between attention scores and inverted average Hamming distances per projection length, computed for 50 randomly selected prompts from GSM8k. Higher projection lengths have stronger correlations.

Projection Length	Mean	Standard Deviation
8	0.2017	0.1890
16	0.2793	0.1852
24	0.3345	0.1806
32	0.3754	0.1792

## D.2 OBSERVATIONS

- **Correlation with Projection Length:** As shown in Figure 8a and Table 8 the average Pearson correlation increases with projection length. This is likely due to the more detailed



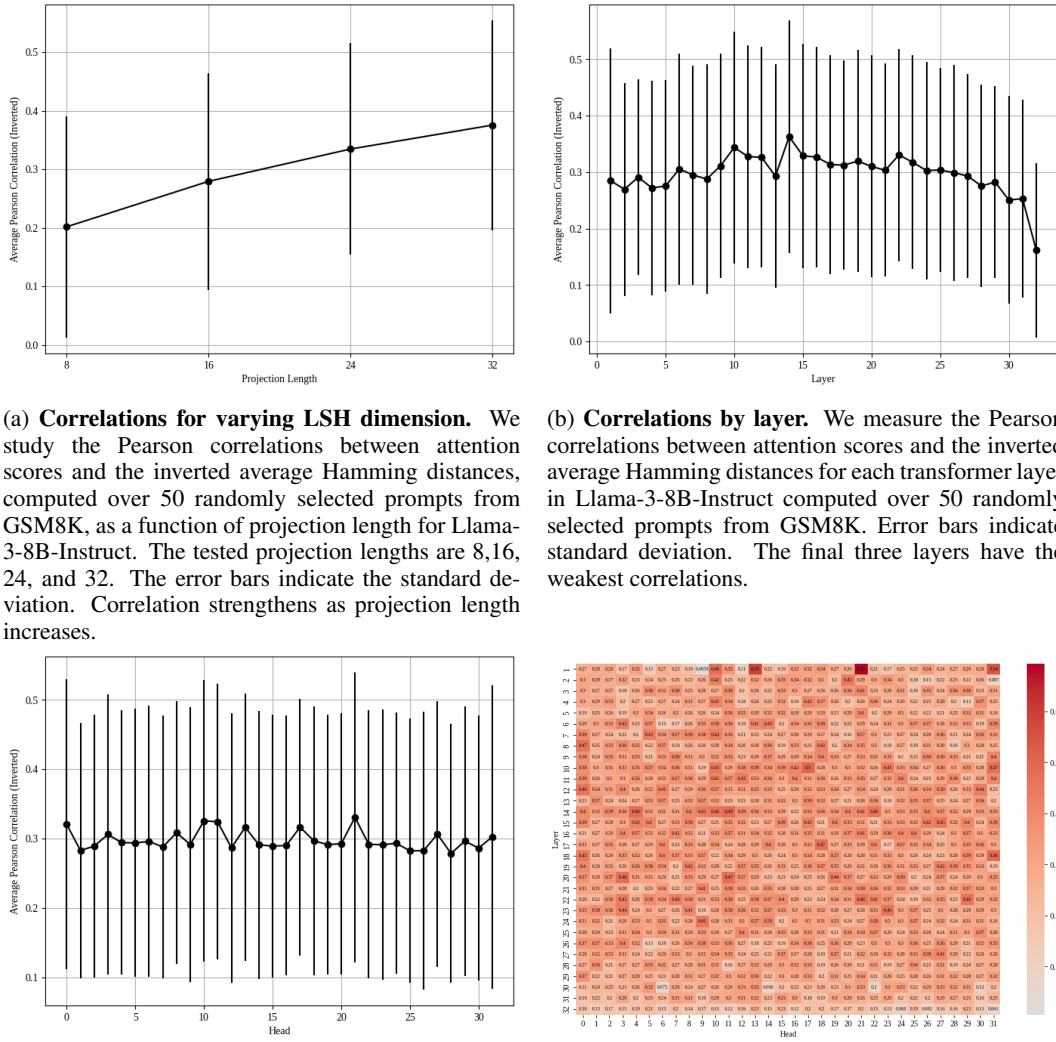


Figure 8: Correlations of Attention and Inverted Hamming Distances

vector representation in the projected space, allowing for finer-grained similarity comparisons.

- **Layer-wise Trends:** Figure 8b shows a slight decrease in the average Pearson correlation for the later transformer layers. Earlier layers may be more focused on recognizing broader patterns where the similarity LSH captures is more pronounced compared to the latter layers, which may focus on specifics not captured as effectively by Hamming distances.
- **Head-wise Consistency:** The correlation between attention scores and inverted average Hamming distance is relatively consistent across different attention heads, with little variance as seen in Figure 8c. This uniform behavior indicates that the relationship between

attention scores and LSH-measured similarity is, to a large extent, independent of specific head functions.

- **LSH vs.  $L_2$  Norms:** While  $L_2$  norms were more effective at identifying high-attention tokens, LSH excelled at identifying tokens with moderate attention scores that are vital for the generation of coherent language output. This aligns with the findings of Guo et al. (2024), which suggests that tokens with low to medium attention scores are crucial for high-quality language generation.
- **LSH and Token Similarity:** LSH tended to group tokens together that are similar across dimensions, producing lower Hamming distances. Tokens with very high attention scores may only have strong associations for a relatively small subset of dimensions, which may not always be captured effectively by LSH.

### D.3 ALR COMPUTATION METHODOLOGY

We compute the Attention Loss Ratio (ALR) for each layer  $l$  and head  $h$  as follows:

1. **Data Capture** During the model’s forward pass, we capture the necessary data for analysis:
  - **Attention Probabilities**  $a_{l,h} \in \mathbb{R}^{n \times n}$ : The attention scores between queries and keys.
  - **Key Norms**  $\|\mathbf{k}_{l,h,p}\|_2$ : The  $L_2$  norms of key vectors at each position  $p$ .
  - **Key and Query Vectors**  $\mathbf{k}_{l,h,p} \in \mathbb{R}^d$  and  $\mathbf{q}_{l,h,p} \in \mathbb{R}^d$ : Used for LSH ranking.
2. **Mean Attention Scores** For each token position  $p$ , we compute the mean attention score across all positions it attends to:

$$\bar{a}_{l,h,p} = \frac{1}{n} \sum_{q=1}^n a_{l,h,p,q}. \quad (8)$$

3. **Ranking Methods**

- **Ideal Attention-Based Ranking** Rank positions in ascending order of  $\bar{a}_{l,h,p}$  (from lowest to highest attention score).
  - **$L_2$  Norm Ranking** Rank positions in descending order of the key norms  $\|\mathbf{k}_{l,h,p}\|_2$ .
  - **LSH Ranking** Apply Locality-Sensitive Hashing (LSH) to key and query vectors using random projections, compute Hamming distances, and rank positions in ascending order of the average Hamming distance.
4. **ALR Calculation** For each  $m$  from 1 to  $n$ , compute the cumulative attention losses: This allows us to quantitatively compare how well different ranking methods (e.g.,  $L_2$  norm and LSH ranking) approximate the ideal scenario where the least important KV pairs (those with the lowest attention scores) are dropped during cache compression.

$$L_{l,h}^m = \sum_{i=1}^m \bar{a}_{l,h,\pi(i)}, \quad (9)$$

$$L_{l,h,\text{ref}}^m = \sum_{i=1}^m \bar{a}_{l,h,\sigma(i)}, \quad (10)$$

where  $\pi(i)$  and  $\sigma(i)$  are the indices of the  $i$ -th position in the ranking method and the ideal attention-based ranking, respectively. The ALR for each head and layer is then calculated as  $Y_{l,h} = \sum_{m=1}^n (L_{l,h}^m - L_{l,h,\text{ref}}^m)$ .

A lower  $Y_{l,h}$  indicates that the ranking method closely approximates the ideal attention-based compression.

5. **Aggregation** We repeat the above steps for multiple prompts and average the ALR values to obtain the final ALR matrix across layers and heads.