

Compactor: Calibrated Query-Agnostic KV Cache Compression with Approximate Leverage Scores

Vivek Chari & Benjamin Van Durme

Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
`{vchari2,vandurme}@jhu.edu`

Abstract

Modern Large Language Models (LLMs) are increasingly trained to support very large context windows. Unfortunately the ability to use long contexts in generation is complicated by the large memory requirement of the KV cache, which scales linearly with the context length. This memory footprint is often the dominant resource bottleneck in real-world deployments, limiting throughput and increasing serving cost. One way to address this is by compressing the KV cache, which can be done either with knowledge of the question being asked (query-aware) or without knowledge of the query (query-agnostic). We present Compactor, a parameter-free, query-agnostic KV compression strategy that uses approximate leverage scores to determine token importance. We show that Compactor can achieve the same performance as competing methods while retaining $1/2$ the tokens in both synthetic and real-world context tasks, with minimal computational overhead. We further introduce a procedure for *context-calibrated compression*, which allows one to infer the maximum compression ratio a given context can support. Using context-calibrated compression, we show that Compactor achieves full KV performance on Longbench while reducing the KV memory burden by 63%, on average. To demonstrate the efficacy and generalizability of our approach, we apply Compactor to 27 synthetic and real-world tasks from RULER and Longbench, with models from both the Qwen 2.5 and Llama 3.1 families.

1 Introduction

Much effort has been devoted to increasing the context length supported by modern Large Language Models (LLMs). While many LLMs now support incredibly long context windows, practical use of long documents is complicated by the autoregressive attention mechanism: at each generation step we must retain all key/value pairs for every past token, so the KV cache grows linearly with context length. For example, running Qwen-2.5 32B on a prompt of size 100K requires maintenance of 26 GB of KV cache, per request; such a memory burden limits the scalability of long-context deployment. This memory burden is even more pressing, given the widespread use of prompt caching to accelerate inference [32, 1, 28].

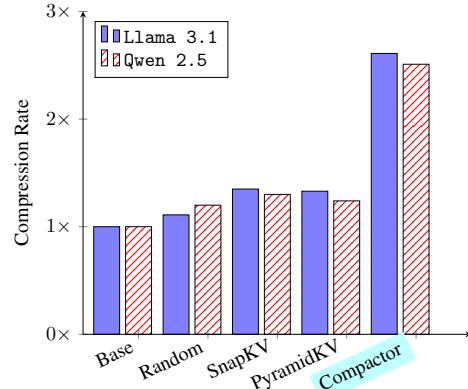


Figure 1: Eviction methods with context-calibrated compression on Longbench. Each maintains similar performance as 100% KV, but at different compression levels.

To address this, two orthogonal lines of work have emerged. One has focused on improving LLM generation efficiency by careful management of the KV cache [22, 41], and designing more complex prompt caching strategies [15]. The other has focused on methods that reduce the memory burden of the KV cache by compressing, evicting, or summarizing tokens. Less work has focused on the integration of the aforementioned techniques.

We identify two key barriers to the integration of KV compression techniques into efficient LLM serving systems. Firstly, most compression schemes rely on knowing the query at compression time; in this "query-aware" setting, the query is used to inform which tokens are evicted. Unfortunately, the performance of query-aware compression methods degrades significantly when the query is not known at compression time (the "query-agnostic" regime) [7, 25]. Additionally, query-aware schemes are incompatible with multi-query prefix caching: each new question would evict different prefix tokens, so the compressed prefix cannot be reused across prompts. Second, even when query-agnostic compression is possible, determining how many tokens can be evicted without degrading quality is non-trivial. Such determinations are especially needed in applications that lack a prescribed token or memory budget, but where generation quality is paramount. To the best of our knowledge, there has been little prior work targeting this issue of ensuring *context-calibrated compression*. Our work is orthogonal, and can be composed with, head-level eviction policies as well [35, 33].

Contributions In this work we present Compactor, a lightweight, parameter-free KV-eviction strategy that operates without query knowledge and still preserves answer quality. Our contributions are as follows:

- (1) We propose Compactor, a two-component eviction mechanism that uses leverage scores and non-causal attention scores to significantly outperform current query-agnostic mechanisms. We further propose a fast randomized algorithm to approximate the aforementioned leverage scores to provide a fast, practical implementation of Compactor.
- (2) We introduce the notion of *context-calibrated compression*, which allows one to infer the maximum compression a particular context can support with minimal inference-time overhead. Our formulation of context-calibration can be applied to any choice of eviction policy. Figure 1 shows that Compactor can achieve significantly higher query-agnostic compression rates than competing methods.
- (3) We evaluate Compactor across multiple LLMs, and both synthetic and real-world long-context datasets. We show that on Longbench Compactor performs comparably to using the full KV cache, while halving the memory burden of the KV cache, with minimal computational overhead.

2 Background

We focus on a Transformer based language model [34]. Applying the standard self-attention mechanism to a context of length N ("prefilling") has time complexity $O(N^2)$, while auto-regressively decoding a single token costs $O(N)$; storing the resulting key-value (KV) cache requires $O(N)$ memory. Prefilling is therefore both compute- and memory-bound, whereas decoding is dominated by memory bandwidth. A large body of work tackles these bottlenecks by manipulating the KV cache at different points in the generation pipeline. To describe this work, we adopt the taxonomy of Li et al. [25], which divides KV cache management into 4 stages: *KV generation*, *KV retrieval*, *KV loading*, and *KV compression*.

KV Generation The aim at this stage is to construct the cache more cheaply during prefilling. Recent work pursues attempts to sparsify attention patterns during prefilling[4, 36, 20]; these works largely differ in the particular sparsity mask they employ. Another line of work tries to replace self-attention altogether with recurrent models that have bounded state sizes [16, 26]. Yet another line of work focuses on cache-aware scheduling during the prefilling stage [13, 40, 9]. Finally, prompt-level compression can be applied *before* the model to drop unimportant tokens, lowering prefill cost proportionally [29]. Such architectural changes are usually query-agnostic.

KV Retrieval & Loading Once a cache exists, the system must decide *which* cached segments to reuse across requests. Efficient inference frameworks already exploit exact prefix matching[22, 41]. When the desired KVs reside in slower memory, they must be loaded to GPU efficiently. For example, PagedAttention in vLLM treats GPU RAM as a pool of fixed-size pages, enabling fine-grained sharing

of KV segments across requests. Similarly, [31] separate the inference workload into prefilling and a decoding cluster, and use a disaggregated KV store to improve (CPU) RAM and SSD utilization. Beyond exact reuse, *semantic* retrieval matches new prompts to a vector database of cached prefixes, by either skipping the prefill when the prefix similarity exceeds a learned threshold [37], or anticipates which KV elements will be attended to and pre-fetching them [23]. Such methods are inherently query-aware.

KV Compression The final line of defense is to reduce the footprint of the cache. Quantization is the most direct route, and reduces the memory footprint of each element in the KV cache [27, 38]. Orthogonal works prune tokens altogether to reduce the sequence dimension. StreamingLLM keeps only a sliding window of recent tokens [36]; H₂O, SnapKV, and PyramidKV use accumulated attention scores to drop the least-attended tokens (see §3.3 for further details) [39, 24, 6]. Variants of the above methods apply different compression ratios or strategies to different heads [33]. Token-dropping techniques work well in the query-aware setting, because the attention that tokens in the question place on tokens in the context are used to select necessary tokens. However, in the query-agnostic regime, their performance suffers greatly [25, 7].

Each of these techniques improves KV-cache efficiency along an orthogonal dimension and can therefore be composed to yield highly efficient inference procedures. In this work we focus on KV-cache compression along the sequence dimension; accordingly, we compare only against methods that explicitly reduce the KV cache size via token-dropping in our experiments.

3 Compactor

Preliminaries & Overview Let a sequence consist of N tokens, each mapped to d -dimensional key and value vectors collected in matrices $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$. Standard self-attention caches the entire (\mathbf{K}, \mathbf{V}) matrices, incurring $O(Nd)$ memory burden. We aim to retain only a fraction $r \in (0, 1]$ of the tokens while preserving a model’s predictive quality. While not an asymptotic improvement in the memory usage, in practice one can achieve significant memory savings with minimal performance loss by utilizing token eviction. To determine which tokens are to be evicted/discarded, we compute a vector $\vec{s} \in \mathbb{R}^N$ of “importance scores” and select the top- k with $k = \lceil r \cdot N \rceil$ highest scoring tokens for retention. To compute \vec{s} , we introduce two complementary score families: geometry-based outlier scores \vec{o} and task-driven attention scores \vec{a} ; each captures different, essential, aspects of the key distribution. We blend them to generate final importance scores \vec{s} .

Notation We consider a transformer-based language model [34], denoted by LM, over a vocabulary \mathcal{V} . Let $c \in \mathcal{V}^N$ denote a context, and with slight abuse of notation, let \tilde{c}_r denote (the KV cache of) a context that has been compressed with retention rate r (i.e. if $r = 0.1$, we retain 10% of tokens). Throughout, indices $i, j \in [N]$ refer to tokens, whereas boldface capitals denote matrices. For a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, A_i is its i -th row, $\text{rank}(\mathbf{A})$ its rank, and \mathbf{A}^\top its transpose. We denote the total, outlier, and attention “importance score” vectors as $\vec{s}, \vec{o}, \vec{a}$ respectively. Let \bar{v} denote the average value of a vector \vec{v} , and $\text{std}(\vec{v})$ denote the standard deviation of the values of the vector. For clarity, we do not notate the head dimension, as the same procedure is applied to heads independently.

3.1 Outlier Scores via Statistical Leverage

We first turn our attention to generating “outlier” scores $\vec{o} \in \mathbb{R}^N$. Retaining merely the most-attended keys fails when a key is rarely queried yet encodes unique information, i.e., it is an *outlier* in the key space. Eviction of these *outliers* leads to irreversible information loss. We formalize the notion of “outlierness” via the classical concept of statistical leverage:

Definition 1 (Leverage Score). *Let $\mathbf{K} \in \mathbb{R}^{N \times d}$ and denote its SVD as $\text{SVD}(\mathbf{K}) = \mathbf{U}\Sigma\mathbf{V}^\top$. Then the leverage score of the i -th row of K*

$$\ell_i = \|U_i\|_2^2 \quad \sum_{i=1}^N \ell_i = \text{rank}(k)$$

Intuitively, rows whose associated left-singular vectors have large Euclidean norm align closely with the directions in which \mathbf{K} has the greatest variance. Omitting these rows would discard significant information. Theorem 1 formalizes the utility of leverage scores in row selection.

Theorem 1 (Spectral Preservation of Leverage Sampling). *Let $\epsilon, \delta \in (0, 1)$ and data matrix $\mathbf{K} \in \mathbb{R}^{N \times d}$ be given and take $k \geq Cd \log(\frac{d}{\delta})\epsilon^{-2}$ for some universal constant C . Construct $\hat{\mathbf{K}}_k \in \mathbb{R}^{k \times d}$ by sampling k times (with replacement) from the distribution that places probability mass ℓ_i/d on the i -th row. Scale each of the k chosen rows by $\sqrt{d/(k\ell_i)}$. Then, with probability $1 - \delta$, we have*

$$(1 - \epsilon)\mathbf{K}^\top \mathbf{K} \preceq \hat{\mathbf{K}}_k^\top \hat{\mathbf{K}}_k \preceq (1 + \epsilon)\mathbf{K}^\top \mathbf{K}$$

where $A \preceq B$ means that $B - A$ is PSD. In practice [5, 30, 21], even deterministically choosing rows corresponding to the top- k leverage scores provides a good spectral sparsification of K .

The above theorem provides an intuitive explanation of why leverage scores are useful in choosing which tokens in the KV cache to retain. As such, we define outlier scores as follows:

$$\mathbf{K} = \mathbf{U}\Sigma\mathbf{V}^\top \quad \vec{\sigma} = \text{diag}(\mathbf{U}^\top \mathbf{U}) = [\ell_1 \quad \ell_2 \quad \dots \quad \ell_n] \quad (1)$$

where $\mathbf{K} = \mathbf{U}\Sigma\mathbf{V}^\top$ denotes the SVD of pre-rope keys. Naively computing leverage scores by computing the SVD of an $n \times d$ matrix $\mathbf{K} = \mathbf{U}\Sigma\mathbf{V}^\top$ is much too slow in practice. Instead we utilize the following simple identity:

$$\text{SVD}(\mathbf{K}^\top \mathbf{K}) = \mathbf{V}\Sigma^2\mathbf{V}^\top \implies \mathbf{U} = \mathbf{K}\mathbf{V}\Sigma^{-1} \quad (2)$$

so we can instead construct \mathbf{U} by computing the SVD of a $d \times d$ matrix and performing two GEMMs. Though this procedure is asymptotically the same complexity, hardware-level optimization of GEMM routines makes Equation 2 significantly faster when $d \ll N$ (as is the case in the KV cache). In the next section, we further reduce the time to compute the leverage scores by approximating them.

3.2 Approximate Leverage Score Computation

Given the computational intensity of computing leverage scores, there has been substantial work in developing methods for approximating them [11, 8, 12, 42]. A key technique in most leverage score approximation algorithms is “sketching” the original matrix to obtain a smaller matrix that roughly preserves the properties of the original, in the following sense:

Definition 2 (Subspace Embedding). *Let $L \subseteq \mathbb{R}^d$ be a linear subspace. Let $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^k$ be a linear map with the property that $\forall x \in L$*

$$(1 - \epsilon)^2 \|x\|_2^2 \leq \|\Phi x\|_2^2 \leq (1 + \epsilon)^2 \|x\|_2^2$$

The map Φ is called a subspace embedding (or “sketch”) for L with embedding dimension $k \leq d$ and distortion $\epsilon > 0$.

We refer to Φ as a “sketching” matrix. In contrast to prior work in leverage-score approximation, which applies left sketches ($\Phi\mathbf{K}$), we apply a right sketch ($\mathbf{K}\Phi$) to reduce the dimensionality of the column-space of the data, at the cost of weaker theoretical bounds on the error in the approximate leverage scores. The reason for using right sketching is three-fold: (1) speeding up computation of Eq 2 requires a right sketch; (2) instantiating a left sketching matrix and subsequently computing $\Phi\mathbf{K}$ requires substantial memory when N is large; and (3) in practice, we observe no degradation in the utility of leverage scores for KV cache approximation when right sketching (see § 5.5). The algorithm to compute approximate leverage scores is presented in Algorithm 1. The quality of the (Gaussian) right-sketching leverage score approximation is formalized in the following theorem:

Theorem 2 (Approximate Leverage Scores). *Let data matrix $\mathbf{K} \in \mathbb{R}^{N \times d}$ and target dimension $k = \lceil r \cdot N \rceil$ be given. Let distortion factor $\epsilon \in (0, 1)$ and failure probability $\delta \in (0, 1)$ be given. Define $\Phi \in \mathbb{R}^{d \times k}$ be a matrix whose entries are drawn i.i.d from $\mathcal{N}(0, \frac{1}{k})$. Compute the SVD of the sketched matrix $\mathbf{K}\Phi$ and approximate leverage scores $\tilde{\ell}_i$ as:*

$$\text{SVD}(\mathbf{K}\Phi) = \tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^\top \quad \tilde{\ell}_i = \|\tilde{\mathbf{U}}_i\|_2^2$$

Algorithm 1 Approximate Leverage Scores

Require: $\mathbf{K} \in \mathbb{R}^{N \times d}$, target dimension k

Ensure: Approximate leverage Scores $\vec{\sigma}$

- 1: $\Phi \in \mathbb{R}^{d \times k}$ with $\Phi_{ij} \sim \mathcal{N}(0, \frac{1}{k})$
 - 2: $\hat{\mathbf{K}} \leftarrow \mathbf{K}\Phi$
 - 3: $\mathbf{G} \leftarrow \hat{\mathbf{K}}^\top \hat{\mathbf{K}}$
 - 4: $\text{SVD}(\mathbf{G}) = \tilde{\mathbf{V}}\tilde{\Sigma}^2\tilde{\mathbf{V}}^\top \quad \triangleright \text{SVD of a } k \times k \text{ matrix}$
 - 5: $\tilde{\mathbf{U}} \leftarrow \tilde{\mathbf{V}}\tilde{\Sigma}^{-1}$
 - 6: **return** $\vec{\sigma} \leftarrow \text{diag}(\tilde{\mathbf{U}}\tilde{\mathbf{U}}^\top)$
-

If $k \geq 12\epsilon^{-2} (\text{rank}(\mathbf{K}) \log(42\epsilon^{-1}) + \log(2\delta^{-1}))$ Then we have that with probability $1 - \delta$

$$\kappa(\mathbf{K})^{-1} \frac{1 - \epsilon}{(1 + \epsilon)} \ell_i \leq \tilde{\ell}_i \leq \kappa(\mathbf{K}) \frac{1 + \epsilon}{1 - \epsilon} \ell_i \quad \forall i \in [N]$$

where $\kappa(\mathbf{K})$ is the condition number of \mathbf{K} .

The proof of the theorem is deferred to Appendix A. Other choices of sketching matrix are also admissible. In Appendix D, we show that choosing Φ to be a Subsampled Randomized Hadamard Transform doesn't impact downstream performance, further demonstrating the robustness of the technique. Furthermore, note that the particular construction of $\tilde{\ell}_i$ shown in Algorithm 1 is not the only one possible; one can also use an economy QR decomposition, or eigendecomposition. We find these also work well in practice. Appendix C discusses other methods for computing approximate leverage scores, which may be more suitable for other hardware. In summary, we apply Algorithm 1 to the (pre-position embedded) key-embeddings to generate outlier scores \vec{o} . With (approximate) outlier scores in hand, we next derive complementary attention-based scores.

3.3 Attention-Based Scoring

While the leverage scores \vec{o} capture the geometry of \mathbf{K} , attention scores capture the task-specific relevance of each token. We now turn our attention to the latter. Using attention-based scores to inform eviction is not a novel proposition. For example H₂O employs the following scoring mechanism, which computes the accumulated attention (over all queries) for each key:

$$\vec{s} = \mathbf{1}^\top \text{softmax}(\mathbf{Q}\mathbf{K}^\top + \mathbf{M}) \quad (3)$$

where $\mathbf{1}^\top \in \mathbb{R}^{1 \times n}$ is a vector of all ones and $\mathbf{M} \in \mathbb{R}^n$ is an upper triangular matrix filled with $-\infty$ to enforce causality, and selects the top- k highest scoring tokens for retention. Similarly, SnapKV uses the attention of the last W queries (on all keys) to generate importance scores:

$$\mathbf{Q}' = [Q_{N-W}^\top \ Q_{N-W+1}^\top \ \dots \ Q_N^\top]^\top \quad \vec{s} = \mathbf{1}^\top \text{softmax}(\mathbf{Q}'\mathbf{K}^\top + \mathbf{M}') \quad (4)$$

where \mathbf{M}' is a causal mask similar to \mathbf{M} with appropriate dimensions. The above mechanism is effective because tokens in W can attend to all prior tokens, and especially effective if the window W contains the query to be answered. However, in the query-agnostic setting there is no reason *a priori* to believe that a window of the last W tokens is more informative than any other window.

Rather than privileging the final W tokens, we desire a query-agnostic statistic that reflects the likelihood that any past token will be attended by any future one. Dropping the causal mask and considering the full attention matrix $\mathbf{A} = \text{softmax}(\mathbf{Q}\mathbf{K}^\top)$ achieves this: each entry A_{ij} measures the model's tendency to route information from position j to position i , independent of generation order. We observe that non-causal attention maps exhibit stable, interpretable features aligned with separator tokens, and few-shot exemplars (see Figure 2 for a representative example; see Appendix B for an extended discussion of this structure). Accordingly, we define the attention-based importance score as the column-wise sum $\vec{a} = \mathbf{1}^\top \text{softmax}(\mathbf{Q}\mathbf{K}^\top)$. Note that computation of the above requires instantiation of the entire attention matrix, which is prohibitively expensive. To avoid this, we chunk the input context into blocks of size C and compute scores over each chunk:

$$\vec{a} = \text{concat}_{i=1}^{\lceil N/C \rceil} \left(\mathbf{1}^\top \text{softmax}(\mathbf{Q}[i]\mathbf{K}[i]^\top) \right) \quad (5)$$

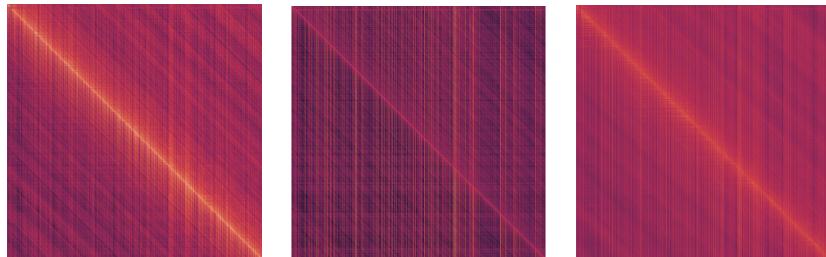


Figure 2: Example of non-causal attention matrices from heads 5, 6, 7 in layer 16 in Llama 3.1-8B Instruct. Lighter colors indicate higher attention scores. Note the columnar and diagonal structure in the non-causal upper-triangular regions of the matrices

where $\mathbf{Q}[i]$ (and similarly $\mathbf{K}[i]$) is the submatrix containing rows $(i-1)C+1$ through iC . We apply mean-pooling to \vec{a} as suggested by [24] and value norm scaling [17] to all mechanisms tested).

3.4 Score Blending and KV Selection

We combine score types into a single ranking signal to inform eviction. We blend the outlier (leverage) scores \vec{o} and attention scores \vec{a} with hyperparameter λ to compute final token-scores as

$$\bar{s} = \frac{\vec{a} - \bar{a}}{\text{std}(\vec{a})} + \lambda \cdot \left(\frac{\vec{o} - \bar{o}}{\text{std}(\vec{o})} \right) \quad (6)$$

The Compactor eviction procedure is then: (1) compute token scores \bar{s} as described by Eq 6; (2) retain the top- $[r \cdot N]$ tokens with the highest scores. Note the scoring and eviction procedure is applied independently to each head and layer.

3.5 Context-Calibrated Compression

The above eviction procedure is parameterized in terms of $r \in (0, 1]$, the fraction of tokens in the KV cache to be retained. If one is given a token budget, the maximum r is fully determined. If we have no token budget a priori, we would like to choose the smallest r that leaves model quality unchanged. However, different contexts can tolerate wildly different compression ratios: a string of UUIDs is minimally compressible, while a context of few-shot examples may permit a higher compression. Table 1 and Figure 4 confirm this observation. To this end, we introduce *context-calibrated compression*, a simple way of inferring the amount of compression (i.e. number of tokens that can be evicted) that can be applied to a given text without performance on downstream tasks.

Let $t \in \mathcal{V}^L$ be any text, and let $\text{LM}(t_i; t_{<i}, c)$ denote the language-model likelihood of the i -th token of t conditioned on the preceding tokens in t and any additional context c . To assess performance degradation, we would like a task-agnostic, scalar measure of how much compression degrades the model’s output. A natural choice is the negative log-likelihood (NLL); note that a higher NLL indicates a more unlikely text. We can compute the token-averaged NLL for a given t as follows

$$\text{NLL}(t; c) = -\frac{1}{L} \sum_{i=1}^L \log(\text{LM}(t_i; t_{<i}, c))$$

Let \tilde{c}_r be the same context after applying a compression technique with retention r . Let q denote any question. Our goal is to predict, for any (q, c, r) pair, the expected NLL ratio:

$$g(r, q, c) = \frac{\text{NLL}(t; q, c)}{\text{NLL}(t; q, \tilde{c}_r)}$$

for any text $t \in \mathcal{V}^L$. To make this possible in the query-agnostic setting, we make the following crucial assumption:

$$g(r, q, c) = \frac{\text{NLL}(t; q, c)}{\text{NLL}(t; q, \tilde{c}_r)} \approx \frac{\text{NLL}(t; c)}{\text{NLL}(t; \tilde{c}_r)} = f(r, c)$$

for some function f , parameterized *only* by c, r ; i.e., we assume the gain in NLL of any text t conditioned on \tilde{c}_r can be explained entirely by the original context and r . Such an assumption is necessary in the query-agnostic regime. Empirically (see Fig. 6) we find $g(r, c)$ decays smoothly and monotonically with r . As such, to approximate $g(r, q, c) \approx f(r, c)$, we fit the two-parameter curve

$$k = \alpha \cdot \text{NLL}(c) + \beta \quad f_{\alpha, \beta}(r, c) = \frac{\exp(rk - k) - \exp(-k)}{1 - \exp(-k)}$$

which satisfies $f_{\alpha, \beta}(1, c) = 1$ (no compression) and $f_{\alpha, \beta}(0, c) = 0$ (empty cache). The free parameters (α, β) are obtained by least-squares regression on training triples

$$(r, c, y) \quad y = g(r, q, c) = \frac{\text{NLL}(t; q, c)}{\text{NLL}(t; q, \tilde{c}_r)}$$

where q is the question being asked, t is the ground truth answer, and y is the ratio of the NLL of the true answer under the original and compressed contexts. At inference time, given a user-specified

quality budget $\tau \in (0, 1]$ and context c , (for example $\tau = 0.95$ allows an NLL increase of at most $\frac{1}{0.95} \approx 5\%$), we choose $r^*(c, \tau) = \arg \max_{r \in (0, 1)} f_{\alpha, \beta}(r, c)$ such that $f_{\alpha, \beta}(r, c) \geq \tau$, for which there is a closed form solution. Compactor then keeps the top $\lceil r^* \cdot N \rceil$ tokens per head. In practice, we use asymmetrical MSE to penalize under-estimation of $r^*(c, \tau)$. In summary, we fit a curve to predict the level of degradation that will occur when retaining r fraction of tokens under a particular compression method; at inference time, we use the inverse of the curve to determine what retention rate the given context can support without excessively degrading performance.

4 Experiments

We demonstrate the efficacy of Compactor on two long-context models: Llama 3.1-8B-Instruct and Qwen 2.5-14B-Instruct-1M (abbreviated as Llama 3.1 and Qwen 2.5 from here). We first conduct benchmarking experiments to demonstrate the overhead cost of running the Compactor scoring mechanism. All benchmarks are conducted on an NVIDIA H100 80GB GPU, and we report median wall-clock time. We assess performance on long-context tasks and degradation across compression ratios via experiments on:

RULER [18] is a synthetic benchmark that asks: *given a stated context window, how much of it can the model actually use?* It consists of 13 tasks organized into four categories: (1) needle-in-a-haystack (NIAH) retrieval with varying numbers of and types of “needles”, (2) multi-hop tracing that forces models to follow chains of references scattered through the text, (3) aggregation tasks that require fusing information from many locations, and (4) stress tests that add distractor tokens to short-form question tasks. We use the RULER-4k variant because (1) tested models demonstrate excellent performance with no compression, (2) models have poor performance at higher compression rates, and (3) we are chiefly concerned with performance as a function of *compression rate*, instead of number of tokens retained.

Longbench [3] is a benchmark designed to evaluate long-context understanding in LLMs. It consists of 21 datasets, covering the following task categories: single-document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion. LongBench requires comprehension of extensive passages, and probes models’ ability to maintain coherence, retrieve relevant information, and reason over non-synthetic long inputs. We evaluate on the English subset, and provide results for tasks that have a well-defined query (which excludes code-completion; for completeness we include results on the coding portion of Longbench in Appendix E)

Additionally, we assess the performance of context-calibrated compression on Longbench under two settings: Zero-Shot, and Finetuned. In the Zero Shot setting, for each document in the Longbench corpus, we predict the maximum compression rate the (eviction method, document) pair supports (see Sec 5.4). In the Finetuned setting, we finetune the base model on documents in the Longbench corpus (not queries or answers), and apply the aforementioned procedure to the finetuned model on the same Longbench corpus. Such an experiment is useful for assessing the compressive capability of the model when targeted for a particular corpus known a priori. The finetuning is performed with rank 128 LoRA adaptors on Q,K,V matrices for 8 epochs [19].

We use the following parameters for all experiments: $\lambda = 0.3$, sketching dimension $k = 64$, chunk size $C = 256$, $\tau = 0.95$ for Llama 3.1 and $\tau = 0.90$ for Qwen 2.5; λ, C were chosen to maximize performance on a validation split of RULER when searching over $\lambda \in \{0.15, 0.2, 0.3, 0.4, 0.5, 0.6\}$ and $C \in \{64, 128, 256, 512\}$. We demonstrate the stability of Compactor across parameter choices in § 5.5. While the choice of $k = 64$ violates the bounds required for Theorem 2, we empirically find the approximation is sufficient to achieve good performance. Such a phenomenon is not a new observation, and much prior work has shown that the bounds tend to be loose on real-world datasets [5, 30, 21]. All experiments are conducted in a query-agnostic manner.

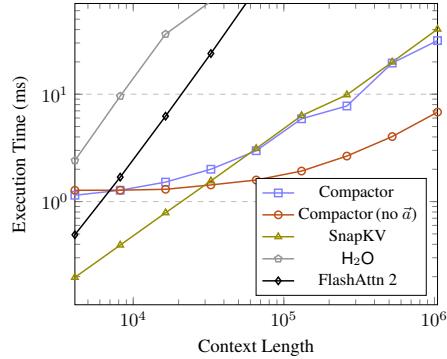


Figure 3: Median wall-clock overhead of the selection mechanism for a single layer of Llama 3.1 (batch size 1).

5 Results

5.1 Selection Mechanism Speed

We benchmark the speed of different eviction mechanisms across various context lengths. Figure 3 shows wall-clock execution time of the eviction mechanism for a single layer for Llama 3.1 (with batch size 1). We plot the execution time of Flash Attention 2 to provide context as a lower bound on the amount of time spent in a layer. We see that Compactor is about as fast as SnapKV for context lengths above 16k tokens; the overhead at short contexts is due to the time spent computing an SVD, but this time is constant for all context lengths.

We additionally show the cost of computing only outlier scores (Compactor (no \vec{a})) to demonstrate the minimal overhead of approximate leverage score computation; the increase in execution time for Compactor (no \vec{a}) is almost entirely due to matrix multiplication (line 5 in Algorithm 1). As such, Compactor can be used as a drop-in replacement over other eviction mechanisms in the query-agnostic regime, with no performance overhead for long contexts.

5.2 RULER

Figure 5 plots mean RULER score (across all tasks) versus KV retention for Compactor and four baselines (SnapKV, H₂O, PyramidKV, random eviction) for both Llama 3.1 and Qwen 2.5. 100% KV retention denotes the score of the model with no compression and is shown as a dashed line. We observe remarkably similar performance characteristics across both models. As the cache is pruned, Compactor degrades most gracefully: at 75% retention it recovers 93.8% of baseline, at 50% it maintains 87.6%, and even at only 10% retention it still achieves 59.5%. Performance when using SnapKV, PyramidKV, or H₂O degrades linearly with KV retention setting, confirming the inadequacy of these methods in the query-agnostic regime. The performance of all methods is poor at severe compression ratios, demonstrating the inherent difficulty of the task. Performance of eviction methods on NIAH and QA tasks is shown in Figure 4, and demonstrates the variable performance that different tasks support.

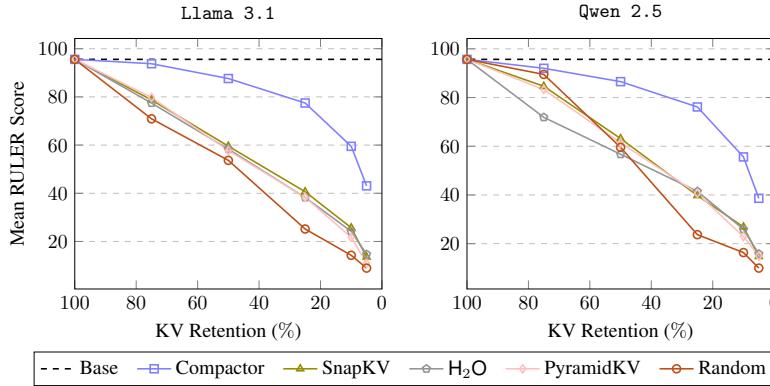


Figure 5: Mean RULER score (all tasks) on Llama 3.1 and Qwen 2.5 across KV retention rates.

5.3 Longbench

We evaluate Compactor on Longbench to assess performance on real-world long context tasks. Table 1 reports per-task and average scores on LongBench for Compactor, SnapKV, and PyramidKV at 50%, 25%, 10%, and 5% retention. H₂O was excluded due to insufficient memory on long contexts. At 50% retention, Compactor matches or slightly exceeds the full-cache performance, with particular gains in few-shot and synthetic tasks with less contextual diversity. Even at 25%, Compactor retains 93% of the full-cache score, whereas SnapKV and PyramidKV retain 74% of the baseline. Under

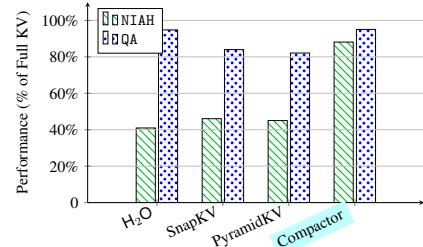


Figure 4: Performance of eviction methods on RULER NIAH and QA subtasks at 50% KV retention cache on Llama 3.1. Performance is measured as a fraction of full KV performance.

Table 1: Longbench scores by meta-task. Cells in blue denote our contribution.

Task	Method	Llama 3.1					Qwen 2.5				
		100%	50%	25%	10%	5%	100%	50%	25%	10%	5%
Single-Doc QA	Compactor	0.485	0.434	0.279	0.172		0.433	0.369	0.277	0.174	
	SnapKV	0.484	0.407	0.290	0.200	0.136	0.438	0.356	0.234	0.184	0.140
	PyramidKV	0.371	0.278	0.186	0.139		0.361	0.230	0.184	0.139	
Multi-Doc QA	Compactor	0.465	0.448	0.389	0.254		0.572	0.549	0.455	0.419	
	SnapKV	0.476	0.407	0.290	0.200	0.136	0.563	0.494	0.423	0.316	0.259
	PyramidKV	0.371	0.278	0.186	0.139		0.490	0.411	0.297	0.264	
Summarization	Compactor	0.283	0.265	0.238	0.206		0.253	0.238	0.214	0.196	
	SnapKV	0.286	0.265	0.242	0.198	0.168	0.258	0.240	0.219	0.182	0.154
	PyramidKV	0.264	0.239	0.197	0.172		0.242	0.212	0.176	0.140	
Few Shot	Compactor	0.509	0.492	0.400	0.364		0.623	0.588	0.497	0.377	
	SnapKV	0.490	0.493	0.441	0.327	0.290	0.617	0.593	0.551	0.482	0.342
	PyramidKV	0.500	0.435	0.321	0.291		0.588	0.545	0.480	0.336	
Synthetic	Compactor	0.594	0.520	0.153	0.081		0.593	0.447	0.153	0.073	
	SnapKV	0.580	0.527	0.382	0.047	0.007	0.607	0.593	0.383	0.127	0.073
	PyramidKV	0.533	0.380	0.094	0.013		0.590	0.377	0.13	0.069	
Total	Compactor	0.458	0.425	0.302	0.225		0.485	0.438	0.331	0.261	
	SnapKV	0.455	0.417	0.344	0.196	0.153	0.489	0.445	0.360	0.267	0.202
	PyramidKV	0.413	0.337	0.201	0.146		0.444	0.353	0.264	0.198	

more aggressive pruning (10% and 5%), Compactor’s advantage grows: it outperforms the next best baseline by over 50% relative at 10%, and by over 40% at 5%. These results confirm that Compactor reliably preserves tokens that drive task performance.

5.4 Context-Calibrated Compression

For each eviction mechanism we fit the context-calibration equation $f_{\alpha, \beta}$, (see §3.5) using tuples of (r, c, y) generated from a subset of RULER. Figure 6 shows the (mean) NLL vs KV retention curves (r vs y) obtained by applying the Compactor eviction mechanism in comparison to SnapKV. Compactor significantly outperforms SnapKV. Note the blue lines along the diagonal of the plot corresponds to the RULER sub-task of UUID needle retrieval, a highly incompressible task; as expected, the NLL gain increases linearly with as KV retention falls. This demonstrates the ability of the proposed calibration function to recognize incompressible text.

We next evaluate the context-calibration curves fit on RULER on Longbench (Figure 7). For each test sample, we first infer the compression ratio r supported by a given context while maintaining a quality budget (NLL ratio) of $\tau = 0.95$ (i.e we accept a $\approx 5\%$ increase in NLL relative to the uncompressed cache). We then retain r fraction of the tokens according to each method’s eviction procedure.

For all tested methods, the performance when applying context-calibrated compression is indistinguishable from full (100% KV) cache performance (one way ANOVA, $p = 0.53$ when performing 20 trials over the randomness in the mechanism). This confirms both the utility of NLL as a proxy

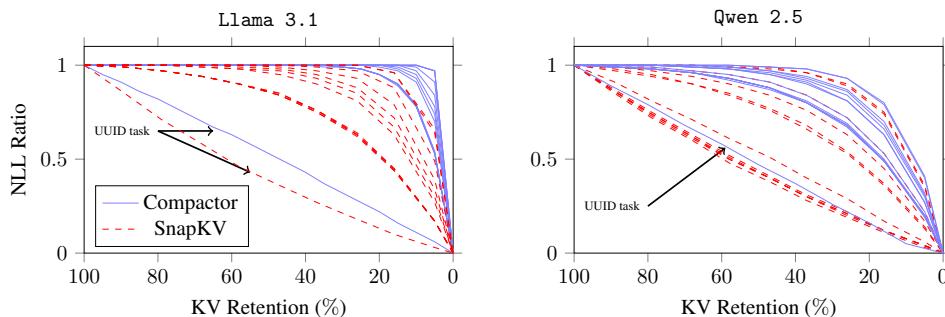


Figure 6: NLL vs KV retention on 13 tasks from RULER, one line per sub-task. For each sub task we compute the (reciprocal) gain in NLL of the ground-truth answer when conditioned on compressed contexts (for each compression method). This empirical trend motivates the exponential calibration function introduced in §3.5

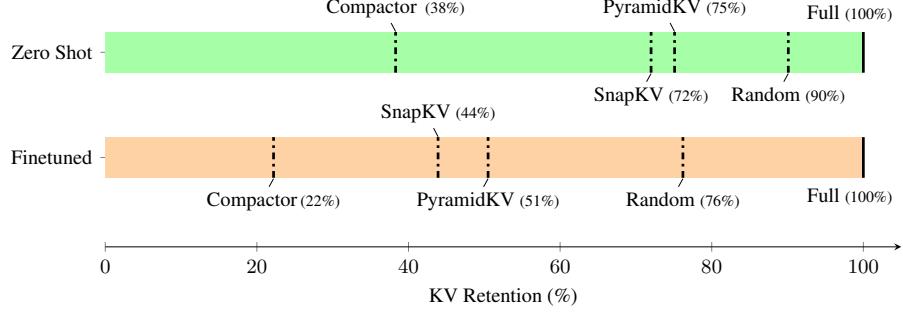


Figure 7: The top bar shows the KV retention rates that each compression method induces when using context-calibrated compression on Longbench. The bottom bar shows the same when the LLM is finetuned on documents from the Longbench test set (no queries). In all cases, the performance of the compression methods is statistically indistinguishable from full KV cache performance. Results are shown for Llama-3.1 8B.

for downstream task performance and the efficacy of the proposed calibration function at ensuring compression does not degrade performance. Note Compactor is able to achieve this performance while retaining significantly fewer tokens ($\approx 37\%$, while SnapKV retains $\approx 72\%$). Both SnapKV and PyramidKV provide only a marginal improvement over the random eviction baseline, far removed from the excellent performance they demonstrate in the query-aware setting [6, 24], where they can match baseline performance while retaining only 10% of the cache.

Lastly, we finetune the model on documents from Longbench (no queries or answers), and evaluate the performance of context-calibrated compression with the finetuned model on Longbench (we evaluate and finetune with the same set of documents). We would expect that the NLL of the documents to fall, allowing for higher compression rates for the documents. Such a procedure is useful when we know *a priori* which corpus will be assessed, and we desire the highest possible accuracy and compression on contexts from this corpus. For example, a question-answering LLM may be finetuned on a corpus of supporting documents, while simultaneously being presented with relevant retrieved documents from that corpus at test time. The result of this experiment is shown in Fig 7 (“Finetuned”). As expected, we see that all models achieve higher compression rates when finetuned on the test corpus. Compactor still significantly outperforms competing techniques, though the gap is shortened; notably, Compactor without training (Zero Shot) outperforms SnapKV and PyramidKV even after finetuning

5.5 Ablations

Finally, we assess the necessity of both components of the scoring mechanism, the stability of the performance under choice of λ , and the result of using the exact leverage scores (Table 2). The addition of attention scores \vec{a} greatly improves the performance of the eviction mechanism, especially at lower compression ratios, while the choice of λ has minimal impact. Usage of approximate leverage scores has little impact on performance, while providing significant speed-ups.

Table 2: Ablation study on RULER of Compactor variants on Llama 3.1. Values represent the mean score across all 13 RULER tasks.

Variant	KV Retention				
	75%	50%	25%	10%	5%
Full ($\lambda = 0.35$)	93.8	87.4	77.5	59.5	43.1
– Exact ℓ_i	93.9	88.1	77.5	59.4	43.0
– $\lambda = 0.15$	93.1	87.8	77.2	60.1	42.1
– $\lambda = 0.2$	93.2	86.4	77.6	58.6	44.1
– $\lambda = 0.4$	92.7	88.2	77.8	58.8	42.2
– No \vec{a}	93.1	81.7	60.2	34.7	24.8

6 Conclusion

We introduce Compactor, a lightweight, *query-agnostic* strategy for KV cache pruning that delivers three concrete benefits over prior work: (1) higher accuracy at higher compression, across both synthetic (RULER) and real-world (LongBench) datasets; (2) Compactor scoring incurs minimal overhead compared to the cost of a single Flash Attention [10] pass for large contexts; and (3) Compactor is parameter-free at inference, hardware-agnostic, and orthogonal to other memory-saving techniques. Our context-calibrated compression rule uses only model likelihoods, which makes Compactor an immediately practical drop-in for long-context LLM serving.

References

- [1] Prompt caching with Claude. URL <https://www.anthropic.com/news/prompt-caching>.
- [2] N. Ailon and B. Chazelle. The Fast Johnson–Lindenstrauss Transform and Approximate Nearest Neighbors. *SIAM Journal on Computing*, 39(1):302–322, Jan. 2009. ISSN 0097-5397. doi: 10.1137/060673096. URL <https://pubs.siam.org/doi/10.1137/060673096>. Publisher: Society for Industrial and Applied Mathematics.
- [3] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, Y. Dong, J. Tang, and J. Li. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL <https://aclanthology.org/2024.acl-long.172/>.
- [4] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The Long-Document Transformer, Dec. 2020. URL <http://arxiv.org/abs/2004.05150>. arXiv:2004.05150 [cs].
- [5] M. Broadbent, M. Brown, and K. Penner. Subset Selection Algorithms: Randomized vs. Deterministic. *SIAM Undergraduate Research Online*, 3:50–71, 2010. ISSN 23277807. doi: 10.1137/09S010435. URL <http://www.siam.org/students/siuro/vol3/S01043.pdf>.
- [6] Z. Cai, Y. Zhang, B. Gao, Y. Liu, T. Liu, K. Lu, W. Xiong, Y. Dong, B. Chang, J. Hu, and W. Xiao. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling, Oct. 2024. URL <http://arxiv.org/abs/2406.02069>. arXiv:2406.02069 [cs].
- [7] V. Chari, G. Qin, and B. V. Durme. KV-Distill: Nearly Lossless Learnable Context Compression for LLMs, Mar. 2025. URL <http://arxiv.org/abs/2503.10337>. arXiv:2503.10337 [cs].
- [8] Y. Chen and Y. Yang. Fast Statistical Leverage Score Approximation in Kernel Ridge Regression. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 2935–2943. PMLR, Mar. 2021. URL <https://proceedings.mlr.press/v130/chen21e.html>. ISSN: 2640-3498.
- [9] M. Cho, M. Rastegari, and D. Naik. KV-Runahead: Scalable Causal LLM Inference by Parallel Key-Value Cache Generation, May 2024. URL <http://arxiv.org/abs/2405.05329>. arXiv:2405.05329 [cs].
- [10] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness, June 2022. URL <http://arxiv.org/abs/2205.14135>. arXiv:2205.14135 [cs].
- [11] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff. Fast approximation of matrix coherence and statistical leverage, Dec. 2012. URL <http://arxiv.org/abs/1109.3843>. arXiv:1109.3843 [cs].
- [12] A. Eshragh, L. Yerbury, A. Nazari, F. Roosta, and M. W. Mahoney. SALSA: Sequential Approximate Leverage-Score Algorithm with Application in Analyzing Big Time Series Data, Dec. 2023. URL <http://arxiv.org/abs/2401.00122>. arXiv:2401.00122 [stat].
- [13] Q. Fu, M. Cho, T. Merth, S. Mehta, M. Rastegari, and M. Najibi. LazyLLM: Dynamic Token Pruning for Efficient Long Context LLM Inference, July 2024. URL <http://arxiv.org/abs/2407.14057>. arXiv:2407.14057 [cs].
- [14] A. C. Gilbert, J. Y. Park, and M. B. Wakin. Sketched SVD: Recovering Spectral Features from Compressive Measurements, Nov. 2012. URL <http://arxiv.org/abs/1211.0361>. arXiv:1211.0361 [cs].
- [15] I. Gim, G. Chen, S.-s. Lee, N. Sarda, A. Khandelwal, and L. Zhong. Prompt Cache: Modular Attention Reuse for Low-Latency Inference, Apr. 2024. URL <http://arxiv.org/abs/2311.04934>. arXiv:2311.04934 [cs].

- [16] A. Gu and T. Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces, May 2024. URL <http://arxiv.org/abs/2312.00752>. arXiv:2312.00752 [cs].
- [17] Z. Guo, H. Kamigaito, and T. Watanabe. Attention Score is not All You Need for Token Importance Indicator in KV Cache Reduction: Value Also Matters. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21158–21166, Miami, Florida, USA, Nov. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1178. URL <https://aclanthology.org/2024.emnlp-main.1178/>.
- [18] C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia, Y. Zhang, and B. Ginsburg. RULER: What’s the Real Context Size of Your Long-Context Language Models?, Aug. 2024. URL <http://arxiv.org/abs/2404.06654>. arXiv:2404.06654 [cs].
- [19] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=nZeVKeFYf9>.
- [20] H. Jiang, Y. Li, C. Zhang, Q. Wu, X. Luo, S. Ahn, Z. Han, A. H. Abdi, D. Li, C.-Y. Lin, Y. Yang, and L. Qiu. MIference 1.0: Accelerating Pre-filling for Long-Context LLMs via Dynamic Sparse Attention, July 2024. URL <http://arxiv.org/abs/2407.02490>. arXiv:2407.02490 [cs] version: 1.
- [21] I. T. Jolliffe. Discarding Variables in a Principal Component Analysis. I: Artificial Data. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 21(2):160–173, 1972. ISSN 0035-9254. doi: 10.2307/2346488. URL <https://www.jstor.org/stable/2346488>. Publisher: [Royal Statistical Society, Oxford University Press].
- [22] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention, Sept. 2023. URL <http://arxiv.org/abs/2309.06180>. arXiv:2309.06180 [cs].
- [23] W. Lee, J. Lee, J. Seo, and J. Sim. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management, June 2024. URL <http://arxiv.org/abs/2406.19707>. arXiv:2406.19707 [cs].
- [24] Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen. SnapKV: LLM Knows What You are Looking for Before Generation, June 2024. URL <http://arxiv.org/abs/2404.14469>. arXiv:2404.14469 [cs].
- [25] Y. Li, H. Jiang, Q. Wu, X. Luo, S. Ahn, C. Zhang, A. H. Abdi, D. Li, J. Gao, Y. Yang, and L. Qiu. SCBench: A KV Cache-Centric Analysis of Long-Context Methods, Mar. 2025. URL <http://arxiv.org/abs/2412.10319>. arXiv:2412.10319 [cs].
- [26] O. Lieber, B. Lenz, H. Bata, G. Cohen, J. Osin, I. Dalmedigos, E. Safahi, S. Meirom, Y. Belinkov, S. Shalev-Shwartz, O. Abend, R. Alon, T. Asida, A. Bergman, R. Glozman, M. Gokhman, A. Manevich, N. Ratner, N. Rozen, E. Shwartz, M. Zusman, and Y. Shoham. Jamba: A Hybrid Transformer-Mamba Language Model, July 2024. URL <http://arxiv.org/abs/2403.19887>. arXiv:2403.19887 [cs].
- [27] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. 2023. doi: 10.13140/RG.2.2.28167.37282. URL <http://arxiv.org/abs/2402.02750>. arXiv:2402.02750 [cs].
- [28] mrbullwinkle. Prompt caching with Azure OpenAI Service - Azure OpenAI. URL <https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/prompt-caching>.
- [29] Z. Pan, Q. Wu, H. Jiang, M. Xia, X. Luo, J. Zhang, Q. Lin, V. Rühle, Y. Yang, C.-Y. Lin, H. V. Zhao, L. Qiu, and D. Zhang. LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression. In L.-W. Ku, A. Martins, and V. Srikanth, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 963–981, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.57. URL <https://aclanthology.org/2024.findings-acl.57>.

- [30] P. Paschou, E. Ziv, E. G. Burchard, S. Choudhry, W. Rodriguez-Cintron, M. W. Mahoney, and P. Drineas. PCA-correlated SNPs for structure identification in worldwide human populations. *PLoS genetics*, 3(9):1672–1686, Sept. 2007. ISSN 1553-7404. doi: 10.1371/journal.pgen.0030160.
- [31] R. Qin, Z. Li, W. He, M. Zhang, Y. Wu, W. Zheng, and X. Xu. Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving, July 2024. URL <http://arxiv.org/abs/2407.00079>. arXiv:2407.00079 [cs].
- [32] T. Sarlos. Improved Approximation Algorithms for Large Matrices via Random Projections. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 143–152, Oct. 2006. doi: 10.1109/FOCS.2006.37. URL <https://ieeexplore.ieee.org/document/4031351>. ISSN: 0272-5428.
- [33] H. Tang, Y. Lin, J. Lin, Q. Han, S. Hong, Y. Yao, and G. Wang. RazorAttention: Efficient KV Cache Compression Through Retrieval Heads, July 2024. URL <http://arxiv.org/abs/2407.15891>. arXiv:2407.15891 [cs].
- [34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need, Aug. 2023. URL <http://arxiv.org/abs/1706.03762>. arXiv:1706.03762 [cs].
- [35] G. Xiao, J. Tang, J. Zuo, J. Guo, S. Yang, H. Tang, Y. Fu, and S. Han. DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads, Oct. 2024. URL <http://arxiv.org/abs/2410.10819>. arXiv:2410.10819 [cs].
- [36] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient Streaming Language Models with Attention Sinks, Apr. 2024. URL <http://arxiv.org/abs/2309.17453>. arXiv:2309.17453 [cs].
- [37] J. Yao, H. Li, Y. Liu, S. Ray, Y. Cheng, Q. Zhang, K. Du, S. Lu, and J. Jiang. CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion, June 2024. URL <http://arxiv.org/abs/2405.16444>. arXiv:2405.16444 [cs].
- [38] T. Zhang, J. Yi, Z. Xu, and A. Shrivastava. KV Cache is 1 Bit Per Channel: Efficient Large Language Model Inference with Coupled Quantization, May 2024. URL <http://arxiv.org/abs/2405.03917>. arXiv:2405.03917 [cs].
- [39] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, Z. Wang, and B. Chen. H\$_2\$O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models, Dec. 2023. URL <http://arxiv.org/abs/2306.14048>. arXiv:2306.14048 [cs].
- [40] Y. Zhao, D. Wu, and J. Wang. ALISA: Accelerating Large Language Model Inference via Sparsity-Aware KV Caching, Mar. 2024. URL <http://arxiv.org/abs/2403.17312>. arXiv:2403.17312 [cs].
- [41] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng. SGLang: Efficient Execution of Structured Language Model Programs, June 2024. URL <http://arxiv.org/abs/2312.07104>. arXiv:2312.07104 [cs].
- [42] Q. Zuo and H. Xiang. A quantum-inspired algorithm for approximating statistical leverage scores, Nov. 2021. URL <http://arxiv.org/abs/2111.08915>. arXiv:2111.08915 [math] version: 1.

A Proof of Theorem 2

Theorem 2. Let data matrix $\mathbf{K} \in \mathbb{R}^{N \times d}$ and target dimension $k = \lceil r \cdot N \rceil$ be given. Let distortion factor $\epsilon \in (0, 1)$ and failure probability $\delta \in (0, 1)$ be given. Define $\Phi \in \mathbb{R}^{d \times k}$ be a matrix whose

entries are drawn i.i.d from $\mathcal{N}(0, \frac{1}{k})$. Compute the SVD of the sketched matrix $\mathbf{K}\Phi$ and approximate leverage scores $\tilde{\ell}_i$ as:

$$\text{SVD}(\mathbf{K}\Phi) = \tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^\top \quad \tilde{\ell}_i = \|\tilde{U}_i\|_2^2$$

If $k \geq 12\epsilon^{-2} (\text{rank}(\mathbf{K}) \log(42\epsilon^{-1}) + \log(2\delta^{-1}))$ Then we have that with probability $1 - \delta$

$$\kappa(\mathbf{K})^{-1} \frac{1-\epsilon}{(1+\epsilon)} \ell_i \leq \tilde{\ell}_i \leq \kappa(\mathbf{K}) \frac{1+\epsilon}{1-\epsilon} \ell_i \quad \forall i \in [N]$$

where $\kappa(\mathbf{K})$ is the condition number of \mathbf{K} .

Proof. The proof of the above is straightforward. It relies on the following intermediary result from [14]:

Corollary 1. Suppose Φ and k are chosen as above, then we have that with probability $1 - \delta$

$$(1-\epsilon) \leq \frac{(\sigma'_j)^2}{\sigma_j^2} \leq (1+\epsilon)$$

where σ_j denote the singular values of \mathbf{K} and σ'_j denote the singular values of $\mathbf{K}\Phi$.

Note that Corollary 1 also applies to the case of right sketching \mathbf{K} . Now consider the following SVDs

$$\mathbf{K}\Phi = \tilde{\mathbf{U}}\tilde{\Sigma}\tilde{\mathbf{V}}^\top \quad \mathbf{K} = \mathbf{U}\Sigma\mathbf{V}^\top$$

Because Φ is a subspace embedding, we have the following relationships between the norms of each row K_i and $(K\Phi)_i$

$$(1-\epsilon)\|K_i\|_2^2 \leq \|(K\Phi)_i\|_2^2 \leq (1+\epsilon)\|K_i\|_2^2$$

Because \mathbf{V} is a unitary transformation, we further have that

$$(1-\epsilon)\|U_i\Sigma\|_2^2 \leq \|\tilde{U}_i\tilde{\Sigma}\|_2^2 \leq (1+\epsilon)\|U_i\Sigma\|_2^2$$

We can define $m = \min_i \Sigma_i^2$ and $M = \min_i \Sigma_i^2$ and $m' = \min_i \tilde{\Sigma}_i^2$ and $M' = \min_i \tilde{\Sigma}_i^2$. It follows that $m' \leq (1-\epsilon)m$ and $M' \leq (1+\epsilon)M$. This implies the following bounds,

$$\begin{aligned} \|\tilde{U}_i\|^2 &\geq \frac{1}{M'} \|\tilde{U}_i\tilde{\Sigma}\|^2 \geq \frac{1-\epsilon}{M'} \|U_i\Sigma\|^2 \geq \frac{1-\epsilon}{M'} m \|U_i\|^2 \\ \|\tilde{U}_i\|^2 &\leq \frac{1}{m'} \|\tilde{U}_i\tilde{\Sigma}\|^2 \leq \frac{1+\epsilon}{m'} \|U_i\Sigma\|^2 \leq \frac{1+\epsilon}{m'} M \|U_i\|^2 \end{aligned}$$

Denote the condition number $\kappa_\Sigma = \frac{M}{m}$, and inserting the above bounds yields the following:

$$\frac{1-\epsilon}{1+\epsilon} \frac{1}{\kappa_\Sigma} \|U_i\|^2 \leq \|\tilde{U}_i\|^2 \leq \frac{1+\epsilon}{1-\epsilon} \kappa_\Sigma \|U_i\|^2$$

which immediately implies the stated bound. \square

B Non-Causal Attention Matrices

In the main text, we used non-causal self-attention (attention without the causal mask) to obtain query-agnostic estimates of how likely any past token will be referenced by future tokens. Empirically, these matrices reveal highly structured patterns that are otherwise suppressed in the causal view. Using these patterns lets us rank tokens by the column-wise sum of their non-causal scores, producing the attention component \mathbf{a} used by Compactor. One feature of the matrices is the presence of “anchor columns”; qualitatively, at these positions, we usually find that separator tokens (‘\n’, ‘,’, ‘. . .’, ‘.’), and conjunctions (and, but) or system prefixes attract disproportionately high attention from many positions, producing bright vertical stripes. We also observe strong locality windows in some heads; even without the mask, attention mass is biased towards a narrow band around the main diagonal, demonstrating the head’s strong local inductive bias.

In Figure 8 we show an example of additional heads from Layer 16, along with the tokens selected by Compactor and SnapKV for these particular heads. This example is taken from the “passage

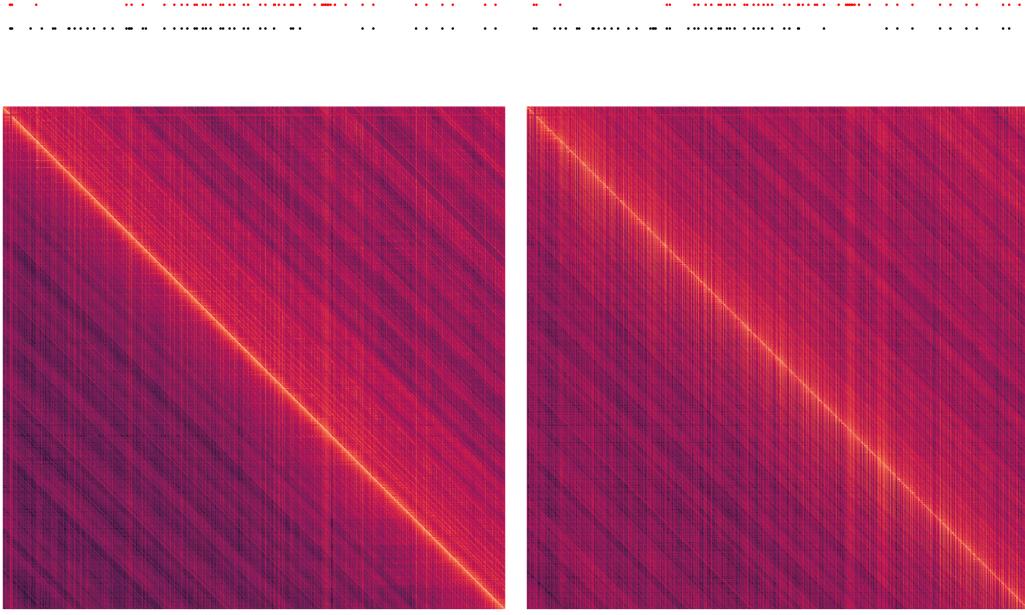


Figure 8: Example of non-causal attention matrices from head 1 and 2 in layer 16 in Llama 3.1-8B Instruct. Lighter colors indicate higher attention scores. Above the matrices, we first include the indices of top 5% of tokens selected by using the non-causal Compactor \vec{a} scores (in red, top row), and then, by SnapKV scores, (in black, bottom row)

retrieval” task on Longbench, in which the model is given 30 paragraphs from Wikipedia, along with an abstract, and must determine which paragraph belongs to the article the abstract is from. In both head 1 and head 2, we observe similar selections in most areas under Compactor and SnapKV. However, in the Compactor selection (upper dots, in red) we observe a tightly clustered region of selected tokens ($\approx 1/3$ of the length from the end) where the paragraph corresponding to the abstract begins. The paragraph is not identified by the SnapKV mechanism (as indicated by the lack of black dots, second line, in that portion of the line) but is identified by Compactor. SnapKV doesn’t get this question correct, while Compactor does. This qualitatively demonstrates the improvement that non-causal attention scores can provide for token selection.

C Subroutines for Computing Approximate Leverage Scores

Throughout the main text we compute an SVD of $\hat{\mathbf{K}} = \mathbf{K}\Phi \in \mathbb{R}^{N \times k}$ to obtain approximate leverage scores. Two potentially cheaper, but less numerically robust alternatives are worth documenting: QR decomposition and the eigendecomposition of the Gram matrix. Both swap a $k \times k$ singular-value decomposition for an alternative operation and therefore (might) admit faster implementations on hardware with optimized primitives; however, they differ in their numerical stability. In this section we assess whether differences in the numerical stability of the methods impact the downstream performance of the methods. We first outline the methods in further detail.

QR Given sketched keys $\hat{\mathbf{K}}$, compute the reduced QR factorization $\hat{\mathbf{K}} = \mathbf{Q}\mathbf{R}$ where $\mathbf{Q} \in \mathbb{R}^{N \times k}$ has orthonormal columns and $\mathbf{R} \in \mathbb{R}^{k \times k}$ is upper-triangular. The leverage score of the i -th token is then

$$\tilde{\ell}_i = \|\mathbf{Q}_{i,:}\|_2^2, \quad i = 1, \dots, N.$$

While QR is backward-stable, the orthogonality of \mathbf{Q} degrades when $\kappa(\hat{\mathbf{K}})$ is large. Reorthogonalization alleviates this but reduces the potential speed-up.

Gram–Matrix Eigendecomposition Form the $k \times k$ Gram matrix $\mathbf{G} = \hat{\mathbf{K}}^\top \hat{\mathbf{K}}$ and compute its eigendecomposition $\mathbf{G} = \mathbf{V}\Lambda\mathbf{V}^\top$, $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_k)$. Then recover an orthonormal left basis via

$$\tilde{\mathbf{U}} = \hat{\mathbf{K}}\mathbf{V}\Lambda^{-1/2}$$

and set $\tilde{\ell}_i = \|\tilde{\mathbf{U}}_{i,:}\|_2^2$. The eigendecomposition is the most sensitive of the three routes. Small singular values of $\hat{\mathbf{K}}$ are squared and may fall below machine precision, inflating the corresponding $\Lambda^{-1/2}$ entries and corrupting $\tilde{\mathbf{U}}$. Clamping $\lambda_j \leftarrow \max(\lambda_j, \varepsilon)$ with $\varepsilon \approx 10^{-6}$ largely mitigates the issue.

Table 3: Compactor RULER scores computed using SVD, QR, and Eigendecomposition on Llama 3.1 8B. Values represent the mean score across all 13 RULER tasks.

Variant	KV Retention				
	75%	50%	25%	10%	5%
SVD	93.8	87.4	77.5	59.5	43.1
QR	93.9	87.2	76.9	59.8	42.9
Eig.	92.1	86.8	76.3	58.2	40.8

Empirically we observe minimal differences between the performance of the SVD and QR variants of Compactor on RULER. As expected, the reduced numerical stability of the eigendecomposition routine results in slightly reduced performance, especially at lower retention rates; however regardless of computation subroutine, Compactor still significantly improves over competing methods (see Table 5). We conclude Compactor is numerically relatively robust to the particulars of the implementation, further confirming the utility of the approximate leverage scores.

D Subsampled Randomized Hadamard Transform

The subsampled randomized Hadamard transform (SRHT) provides an alternative right-sketching matrix $\Phi \in \mathbb{R}^{d \times k}$ that combines three operations:

$$\Phi = \sqrt{\frac{d}{k}} \mathbf{R} \mathbf{H} \mathbf{D}, \quad \mathbf{H} \in \{-1, 1\}^{d \times d}.$$

where \mathbf{D} is a diagonal matrix whose entries are i.i.d. Rademacher (± 1) variables, \mathbf{H} is the Walsh–Hadamard matrix (which can be applied with the fast Hadamard transform in time $O(d \log d)$), and \mathbf{R} selects k columns uniformly without replacement. The above construction of Φ satisfies the same bounds in Theorem 2 as the Gaussian sketch [2]. The key advantage of utilizing a SRHT is that it can be computed in-place on the KV cache matrices, which can reduce peak memory usage. Empirically we observe that substituting the SRHT for the Gaussian right sketch (all else equal) yields negligible differences in performance on RULER with Llama-3.1-8B. In conjunction with Appendix C, these results further demonstrate that Compactor is relatively robust to implementation details.

Table 4: Compactor RULER scores computed using Gaussian sketch and SRHT on Llama 3.1 8B. Values represent the mean score across all 13 RULER tasks.

Variant	KV Retention				
	75%	50%	25%	10%	5%
Gaussian	93.8	87.4	77.5	59.5	43.1
SRHT	93.7	87.8	77.4	59.9	42.4

E Longbench Code Completion

As mentioned in the main body of the paper, we exclude Longbench code-completion tasks from evaluation with other query-agnostic tasks because of the lack of a defined query. For completeness, we include results on the code-portion of Longbench in Table 5. We observe that Compactor achieves close to baseline performance on Llama 3.1, despite being evaluated in a query-aware setting.

SnapKV demonstrates the best performance by a small margin, showing the utility of the recent token window for code completion tasks. Such an observation is not surprising: the code to be completed disproportionately relies on the last window of text to determine what exactly will be completed. The increase in performance of all methods under compression for Qwen 2.5 suggests that the entire context is not needed for the task at hand.

Table 5: Longbench scores for code completion. Cells in blue denote our contribution.

Task	Method	Llama 3.1					Qwen 2.5				
		100%	50%	25%	10%	5%	100%	50%	25%	10%	5%
Code Completion	Compactor	0.250	0.248	0.231	0.229		0.263	0.281	0.292	0.244	
	SnapKV	0.253	0.255	0.249	0.256	0.223	0.255	0.275	0.290	0.288	0.259
	PyramidKV	0.247	0.254	0.224	0.224		0.270	0.289	0.272	0.233	