Exploiting Sparsity for Long Context Inference: Million Token Contexts on Commodity GPUs

Ryan Synk [*1] **Monte Hoover** [*1]
John Kirchenbauer [1] **Neel Jain** [1] **Alex Stein** [1] **Manli Shu** [2] **Josue Melendez Sanchez** [1]
Ramani Duraiswami [1] **Tom Goldstein** [1]

Abstract

There is growing demand for performing inference with hundreds of thousands of input tokens on trained transformer models. Inference at this extreme scale demands significant computational resources, hindering the application of transformers at long contexts on commodity (i.e not data center scale) hardware. To address the inference time costs associated with running self-attention based transformer language models on long contexts and enable their adoption on widely available hardware, we propose a tunable mechanism that reduces the cost of the forward pass by attending to only the most relevant tokens at every generation step using a top-k selection mechanism. We showcase the efficiency gains afforded by our method by performing inference on context windows up to 1M tokens using approximately 16GB of GPU RAM. Our experiments reveal that models are capable of handling the sparsity induced by the reduced number of keys and values. By attending to less than 2% of input tokens, we achieve over 95% of model performance on common benchmarks (RULER, AlpacaEval, and Open LLM Leaderboard). The code is available at https://github.com/ryansynk/topk-decoding.

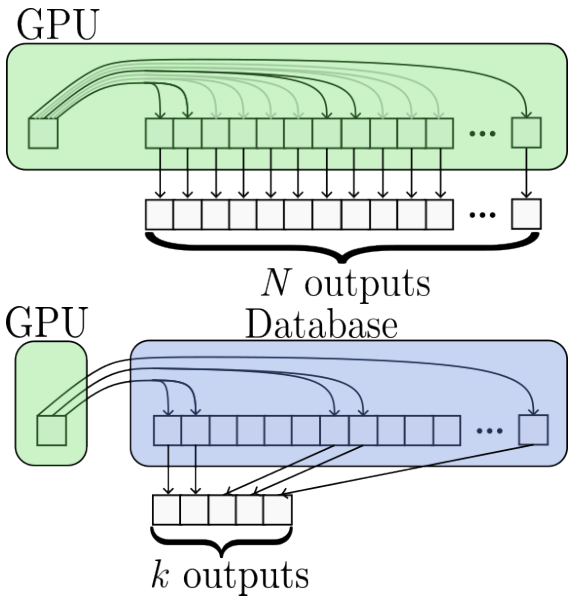


Figure 1: (top) Typical attention requires each query vector to compute an inner product with each key vector in the context window. In practice, most key vectors produce insignificant attention scores, and therefore contribute very little to subsequent hidden states, so much of this computation is wasted. (left-bottom) top-k attention retrieves only the keys that contribute significantly to the attention computation, leaving the gray arrows out and achieving sublinear runtime.

1. Introduction

Long context inference is the process by which models analyze large document collections or follow long and detailed instructions. The *context* is a series of tokens, like a set of documents or a set of files in a codebase. Increasingly, models are trained to handle larger and larger context lengths, with new models training on contexts in the millions (Liu et al., 2024b).

Inference is done using a trained model and is divided into two steps: *prefill* and *decoding*. In the prefill stage, tokens pass forward through the model, and their key and value activations from the attention mechanism of each layer are stored in a list. This list is called the *KV cache* (Pope et al., 2023). In the decoding stage, the model uses the KV cache to generate new tokens one at a time. In a standard attention mechanism, each new token attends to every cached token in the context. When the context length is very large, both the prefill and decoding stages can become prohibitively expensive.

[*]Equal contribution [1]University of Maryland, College Park [2]Salesforce Research. Correspondence to: Monte Hoover <mhoover4@umd.edu>, Ryan Synk <ryansynk@umd.edu>.

The prefill stage incurs an $O(N^2)$ compute and memory cost due to the large attention matrix formed in the computation (where N is the number of tokens in the context). To mitigate this, brute-force methods such as Ring Attention (Liu et al., 2023) were developed. This approach uses round-robin communication between servers to scale computation linearly in the number of GPUs. Despite its apparent effectiveness, while the prefill stage occurs only once, because the decoding stage occurs many hundreds of times for a single user request, the algorithm limited in its practicality outside of the data-center.

Every step during the decoding stage incurs an $O(N)$ compute and memory cost, as each new token must attend over all previous tokens in the cached context. As context lengths grow, the KV cache can become prohibitively large. If each of the N tokens in the context is embedded into a vector of size D (the *hidden dimension*) then the number of floating point numbers in the cache is $2NDL$, where L is the number of network layers. For a value of $D = 4096$ and $L = 32$ as in the Llama-3 8B architecture (Grattafiori et al., 2024), at $N = 100,000$ the memory required for the cache alone is 52GB, assuming 16 bit floating point format (Zhang et al., 2024). At this context length the cache is unable to fit on most commodity GPUs.

One approach to save memory during decoding is to offload the KV cache to CPU memory (Sheng et al., 2023). However, the additional data movement required to implement this strategy is itself prohibitive. In the above example, just a single layer's worth of KV cache data is 1.6GB and this payload must be schlepped back and forth hundreds or thousands of times during a single generation. Attacking the problem from a different angle, cache eviction methods try to maintain a fixed cache size on the GPU by selectively removing token vectors deemed irrelevant (Zhang et al., 2023), but because it can be difficult to tell which vectors will be needed in the future, these strategies ultimately hurt performance.

In contrast to the compute and communication intensive solutions discussed thus far, we base our proposal on a simple fact:

Modern LLMs only need to pay attention to a handful of tokens at a time.

We exploit this observation to perform fast long context decoding with very little GPU memory. We build an implementation of attention in which keys and values are stored in a vector database in CPU memory. When attention is computed using a query vector at decoding time, we retrieve only the $\texttt{k}$ keys with the highest attention scores. This can be done in sublinear time using an approximate $\texttt{k}$-nearest neighbor search over the cache database, enabling long context inference using plentiful and cheap CPU memory, and without the heavy computational overhead required for full attention. Due to the fact that the constraint on the number of relevant keys constrains the CPU-GPU transfer volume, our approach incurs minimal data movement costs compared to other KV cache offloading methods since only a fraction of the token representations ever need to be transported between GPU and CPU memory.

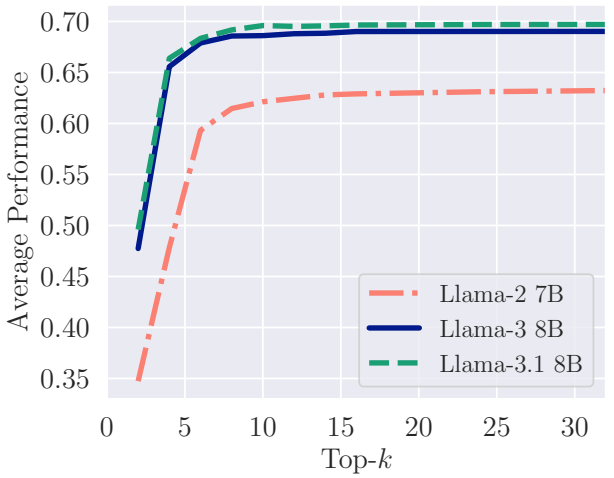


Figure 2: Performance on selected OpenLLM Leaderboard tasks using only the top-$\texttt{k}$ keys for each attention computation. Typical questions have a context length of ~ 1000, yet only 10 keys are needed to achieve the same performance as full attention. We evaluate an extended set of models in Figure 4.

To build a deeper understanding of why our approach works, through an array of controlled experiments we interrogate the core "top-$\texttt{k}$" assumption on which we base our method; that attention in modern LLMs is an inherently sparse operation. While methods exploring the restriction of the attention mechanism to the top-$\texttt{k}$ scores have been studied extensively (Gupta et al., 2021; Liu et al., 2024a; Zhang et al., 2024), prior work does not specifically endeavor to solve resource constrained long-context decoding using scalable tools like approximate nearest neighbor search, nor do they demonstrate million token context inference on single, commodity GPUs.

Our primary contributions are summarized as follows:

- We propose a simple method for sublinear long context generation using top-$\texttt{k}$ attention over preprocessed states in a vector database.
- We show that this technique achieves high fidelity on common benchmarks and long-context tasks (Figure 2).
- We provide an empirical analysis on why top-$\texttt{k}$ attention works and demonstrate its effectiveness at the million token scale.
- We demonstrate the flexibility of our method by varying the top-$\texttt{k}$ budget on a per-layer basis.

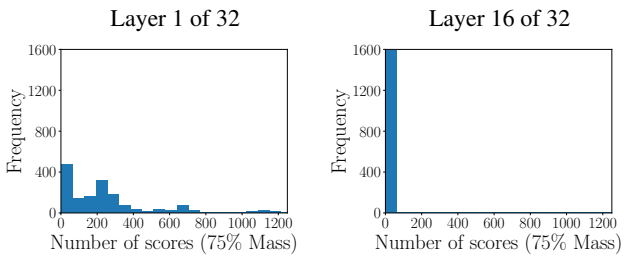


Figure 3: We analyze the number of attention scores that correspond to 75% of the probability mass for generating the next token. Each point is the number of scores of the last 'row' from the attention matrix required to reach 75% of the total attention. We observe each of the 32 heads across 50 samples. On the **left**, we plot the histogram for the first layer in the network, and on the **right** we plot it for layer 16.

2. Motivation

The core assumption underpinning our proposal is the fact that modern language models naturally exhibit sparse attention patterns in which a very small number of tokens make up the vast majority of attention mass. We begin by verifying that this is in fact true for popular models in simple settings.

Observing attention sparsity in the wild. In Figure 3, we analyze 50 4000-token text samples consisting of concatenated Wikipedia article snippets. We encode these samples using a forward pass through Llama-3-8B, and visualize the resulting attention scores. For the last token in each context window, we tabulate the number of keys in the context that are needed to collect 75% of the attention mass. Only a small number of the 4000 tokens are needed to collect this mass, especially for deeper layers of the networks.

The observations made in this series of small scale experiments together suggest that only a few key-value pairs should be needed for the model to perform close to its full capabilities. To demonstrate that the observed sparsity can actually be leveraged without compromising performance, we perform a more realistic test.

We evaluate models from the Llama family on small selection of knowledge-intensive benchmark tasks while constraining the attention computation for each query to only use the top-k key-value pairs with the highest attention score (see Section 3.2 for algorithmic details). In these experiments, we use the same number of keys for each layer of the network and average the model performance over all tasks on the OpenLLM leaderboard (see Section 4.1 for task description). In Figure 2 we see that all three models saturate in performance by the 15th key, and the most recent (and most overtrained) variants of the model require only the 10 top keys to achieve the same performance as full-scale attentions. While Figure 3 indicates that there is some spreading of the attention mass for layer 1 of the network, this tail mass seems to be unnecessary for good performance on benchmarks.

Taken as a whole, these simple but motivating experiments evidence the basic sparsity assumptions on which our method is based. Empirically, we find that the attention scores are a native, reliable indicator as to which key-value pairs are most critical during any given inference step, and that very few keys are actually needed to perform accurate inference.

In the next section we introduce the technical details of our approach for using a vector database to retrieve the top-k most influential tokens, thereby disconnecting attention's selection operations from the rest of the feed-forward computation. This critical separation enables the GPU to perform smaller matrix operations that fit into its limited memory whilst the full cached keys and values live in CPU memory and are searched using CPU compute. In the main experiments that follow, we showcase how our method alleviates both the computational and memory barriers that normally bottleneck long-context inference and unlocks million token contexts for small GPUs.

3. Methodology

3.1. Inference With KV Caches

As stated previously, causal inference using trained transformer models is split into two steps: the prefill and decode stages. Assume we are given a—potentially very large—set of tokens x of size N that we would like to make many queries over.

The query, key and value embeddings during prefill are of size $(N \times D)$, so this stage of inference incurs a maximum activation memory cost of $O(N^2)$ due to the size of the attention matrix. Memory saving attention methods like (Dao et al., 2022) can in practice reduce this to an $O(N)$ memory cost by tiling the attention computation. As the cache is being filled, it also grows in size and persistently costs $O(N)$ memory across the duration of the computation.

In the decoding stage, we have a smaller query we would like to make over this context, as well as a pre-filled KV cache. Our queries comes in one token at a time, with decoding in a self-attention layer performed as follows:

$$\text{Attention}(q, K, V) = \text{Softmax}\left(\frac{qK^T}{\sqrt{D}}\right)V \qquad (1)$$

Here $q \in \mathbb{R}^{1\times D}$ is the new query vector, , $K, V \in \mathbb{R}^{N\times D}$

are the KV cache for that layer, and D is the hidden dimension. We refer to $S = qK^T \in \mathbb{R}^{1\times N}$ as the *score* matrix, and the quantity Softmax $\left(\frac{S}{\sqrt{D}}\right)$ as the *attention* matrix.

When the incoming query q is multiplied by all previous keys in the cache during decoding, the memory and compute cost incurred for the score matrix at this step is $O(N)$. This dependence on N for the cost of GPU memory during inference can become prohibitive for large context lengths.

3.2. Top-k Attention: Accelerating Decoding

We cut down on the growing memory and compute costs by dynamically considering only the most relevant keys in the KV cache. To do this we perform a `k`-nearest neighbor search over the key vectors with the new query vector, returning only those value vectors corresponding to the `k`-largest score values.

The core of our implementation is in Algorithm 1. We assume we are given a prefilled KV cache that is on the CPU. This cache is a sequence $\{K_\ell, V_\ell\}_{\ell=1}^{L}$ of key and value activation tensors, one for each layer.

First we convert the key tensors across all layers in the KV cache into a nearest-neighbor search index. This index data structure can be as simple as a list (for exact nearest neighbors) or as sophisticated as a graph-based structure (for approximate nearest neighbors) as in (Malkov & Yashunin, 2020). For multi-head attention, we construct a separate index for each key head at every layer. For each layer we store the indices in a list called `K_index`.

We embed each new token during decoding into q, k, and v on the GPU. We offload q to the CPU and perform a nearest neighbor search to get the top-`k` score values of the query over the context:

$$\texttt{vals}, I = \texttt{knn\_search}\left(q, \texttt{K\_index}[\ell], \texttt{k}\right)$$

Here `k` is the number of the largest score values we'd like use. The operation `knn_search` performs a `k`-nearest neighbor search of q over the key vector index for layer ℓ, `K_index`$[\ell]$. Here the `k`-nearest neighbor search distance is measured using the dot product metric, mirroring the attention score mechanism. This search returns two items: `vals`, a vector of size `k` containing the top-`k` score values, and I, a list of `k` integers mapping the score values in `vals` to their corresponding key vectors. Let $I = (i_1, \dots i_k)$. We collect the relevant keys selected by the nearest neighbor search by concatenating them as row vectors together, denoted as $V_\ell[I]$:

$$V_\ell[I] = \begin{pmatrix} — & V_\ell[i_1] & — \\ — & V_\ell[i_2] & — \\ & \vdots & \\ — & V_\ell[i_k] & — \end{pmatrix} \in \mathbb{R}^{k\times D}$$

After the model generates the first token using the context, the key and value embeddings of this token are left on the GPU directly. This splits the total KV cache into two parts: a large part on the CPU constructed from the given context, and a small part on the GPU constructed from previously generated tokens. Given a new query, the `k`-nearest neighbor search is only performed on the CPU cache; on the GPU, we compute the attention directly between this query and the keys from previous generated tokens. This process resembles windowed attention (Beltagy et al., 2020a), where a window of recently generated tokens are stored directly on the GPU. These GPU window caches are labeled `K_gen` and `V_gen` in Algorithm 1.

The $V_\ell[I]$ and `vals` are then moved to the GPU. Using these, we perform the final attention computation, combining both the weighted values from the context as well as the value vectors from previously generated tokens. This method gives us a peak GPU memory cost of $O(k)$ GPU memory cost, as opposed to $O(N)$. We find that k can be chosen to be a small fraction (around 1% for most tasks) of N while still recovering near-equivalent performance.

Algorithm 1 Top-`k` KV Cache Decoding

Require: KV-Cache $\{K_\ell, V_\ell\}_{\ell=1}^{L}$ where $K_\ell, V_\ell \in \mathbb{R}^{N\times D}$, token $x \in \mathbb{R}^{1\times D}$, $\texttt{k} \in \mathbb{N}$

$N_{gen} = 1$, `K_gen` $= []$, `V_gen` $= []$
`K_index` = []
for $\ell \in \{1, \dots, L\}$ **do**
  `K_index`$[\ell] \leftarrow$ `build_index`(K_ℓ)
end for
while $N_{gen} < N_{max}$ **do**
  for $\ell \in \{1, \dots, L\}$ **do**
    Pre-attention transformer layer computations...
    $q = xW_{Q_\ell}, k = xW_{K_\ell}, v = xW_{V_\ell}$
    `K_gen`$[\ell] \leftarrow$ `concat`(`K_gen`$[\ell], k$)
    `V_gen`$[\ell] \leftarrow$ `concat`(`V_gen`$[\ell], k$)
    `vals`$, I \leftarrow$ `knn_search`$(q,$ `K_index`$[\ell],$ `k`$)$
    Move $V_\ell[I]$, `vals` to GPU
    $\hat{x} \leftarrow$ `Softmax`$(\frac{1}{\sqrt{D}}$`vals`$)V_\ell[I]$
    $\hat{x} \leftarrow \hat{x} +$ `Softmax` $\left(\frac{1}{\sqrt{D}} q \cdot \texttt{K\_gen}[\ell]^T\right)$
    Post-attention transformer layer computations...
  end for
  $x \leftarrow$ `sample_new_token`$(\hat{x})$
  $N_{gen} \leftarrow N_{gen} + 1$
end while

3.3. Prefilling a KV Cache at the Million Token Scale

The prefill forward pass in the construction of a KV cache potentially requires incurring the full $O(N^2)$ memory cost. However, there are multiple ways one could prefill a cache.

Given (relatively brief) access to large amounts of com-

pute, one could parallelize over many GPUs and construct the cache using algorithms like Ring Attention (Liu et al., 2023). The larger up-front cost of the cache construction would then be amortized over the many queries that would be made over it on much cheaper hardware. Additionally, the attention could be approximated in the construction of the cache. Algorithms like windowed attention would allow for the construction of large caches with more modest compute as in (Child et al., 2019). For small enough N (100's of thousands of tokens), and with a high-memory GPU, the vLLM library can quickly construct KV caches by performing a standard forward pass on the model using the paged attention algorithm (Kwon et al., 2023). Finally, for very large N, top-k attention could be employed at cache construction time as well.

For our largest experiments, we utilize Flash Attention and an H100 GPU to prefill the caches N=1M tokens of context. This allows us to generate exact prefilled KV caches with only small modifications to the network to accommodate the extreme memory requirements. The exact changes we made in order to prefill caches can be found in the Appendix and we employ a chunking strategy similar to (Gupta et al., 2021). We note that our experiments mimic an important use case of our method, namely a user with a large amount of documents that has access to only a limited amount of compute. In this case, the user can rent the required compute on the cloud for prefill *once*, then make as many queries as they want quickly with our method on their own hardware.

4. Evaluating Top-k Attention at Scale

We evaluate top-k to highlight the relationship between different values of k and performance. We find that models of various sizes and generations perform well even at small k. In general, we observe that a k equal to 2% of the total length of the context is sufficient to achieve 95% of the performance achieved with full, standard attention.

4.1. Benchmark Details

We analyze the effectiveness of top-k attention across three benchmarks: RULER, Open LLM Leaderboard v1, and AlpacaEval. Each of these benchmarks highlight a different aspect of top-k attention impact on LLM performance. RULER specifically tests long context abilities and we use Open LLM Leaderboard v1 and AlpacaEval together to examine how different model sizes and families perform under top-k attention. Measuring performance across these benchmarks presents a comprehensive understanding of how the top-k attention mechanism affects a model.

RULER To demonstrate that top-k attention with small k remains effective as the context length increases, we run the RULER (Hsieh et al., 2024) benchmark suite over a series of increasing context lengths. As shown in Table 1 we run over lengths from 4k to 128k. RULER consists of thirteen total tasks from four categories. The evaluation harness runs the original Needle In A Haystack (NIAH) (Kamradt, 2023) task along with a series of more challenging variations of the task (for example, in one task the text consists entirely of labeled "needles" and the model is queries to retrieve the needle corresponding to a single label.) These NIAH tasks comprise 8 of the 13 tasks, and the remaining tasks are split into three categories: summarization proxies, multi-hop proxies, and question answering.

Open LLM Leaderboard Tasks We investigate the performance of top-k on Open LLM Leaderboard tasks. In particular, we evaluate different models on various values of k on the average of MMLU (Hendrycks et al., 2020), ARC tasks both easy and challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), winogrande (Keisuke et al., 2019), OpenbookQA (Mihaylov et al., 2018), BoolQ (Clark et al., 2019), and PiQA (Bisk et al., 2020). For each task, we record the normalized accuracy when available; otherwise, we record accuracy. We report the average over tasks. We evaluate the following models on these benchmarks: Llama-1 (7B), Llama-2 (7B), Llama-3 (8B), Llama-3.1 (8B), Vicuna-v1.3 (7B), Llama-2 chat (7B), Llama-3 Instruct (8B), Llama-3.1 instruct (8B), Llama-3.2 1B instruct, and Llama-3.2 3B instruct (Touvron et al., 2023a; Zheng et al., 2023; Touvron et al., 2023b; Dubey et al., 2024; AI, 2024). The experiments are conducted using `lm-eval-harness` in a zero-shot setting (Gao et al., 2024).

AlpacaEval 2.0 We benchmark top-k attention on AlpacaEval (Dubois et al., 2024). AlpacaEval 2.0 requires a model to generate responses to 805 queries. These responses are then compared by an LLM-as-a-Judge with GPT-4 Turbo responses generated from the same query set. The winrate percentage is the reported metric, and the LLM-as-a-Judge is GPT-4 Turbo.

4.2. Evaluation Results

Top-k Performance on RULER We evaluate our method on RULER using GradientAI's Llama-3-8B model that was trained using a context length of 262k tokens. For this model, we find that very small values of k are sufficient to recover near-baseline performance. At every context length evaluated, 95% of the baseline performance can always be achieved with a k value of 1% or less of the total length. In Table 1, at $k = 2$, greater than 60% performance is achieved at all context lengths. The performance on RULER improves as k increases. Nevertheless, even at a context length of 131k tokens to achieve $\sim$ 98% performance of the full attention only 12.5% of the attention scores are required.

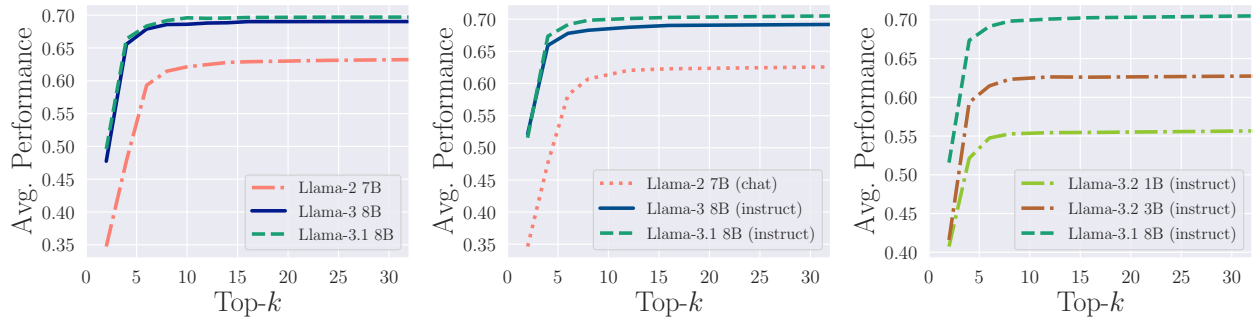


Figure 4: Top-k attention is effective for OpenLLM Leaderboard Tasks even at small values of k. Left shows the average of all tasks as we increase k on pretrained base models. Center shows instruction tuned models. Right investigates the performance on different model sizes.

Interestingly, NIAH tasks have high success rates with little variation across choice of k. Question-answering tasks (from SQuAD (Rajpurkar et al., 2016) and HotpotQA (Yang et al., 2018)) have lower success rates with the 8B-size model, but also show low variation in success rates across different k. In our experiments, word-counting tasks (CWE and FWE) were the tasks most affected by the choice of k. This suggests that in practice, extra compute (larger k) should be allocated to these types of tasks to maintain performance. Table 2 shows the differences between each of these tasks in terms of what k-value is required to reach near-parity performance.

Top-k Performance on OpenLLM Leaderboard Tasks We evaluate various models to find that top-k behavior exists regardless of instruction tuning, model size, or the number of tokens the model was trained on. Figure 4 left shows that regardless of the number of tokens on which the model was trained, all exhibit a similar trend with performance saturating at k values < 10. When comparing Figure 4 left and center, we see that instruction models and pretrained base models exhibit similar behavior, saturating very quickly. Finally, in Figure 4 right, while different models achieve different performance maximums on the benchmark suite, we see that the effect of top-k is independent of model size.

Table 1: Results on RULER benchmark at various context lengths. Scores represent an average of 13 tasks in the RULER benchmark, with maximum possible score being 100. The RULER benchmark was run separately for each context length listed, and each context length was run with top-k attention for increasing values of k and also with standard, full attention.

	Context Length (tokens)				
k	8192	16384	32768	65536	131072
2	70.58	71.27	68.00	67.83	64.76
8	83.10	86.69	84.92	75.99	61.28
32	85.38	88.08	85.20	78.18	65.89
128	87.21	89.08	85.16	77.41	73.59
512	88.55	89.31	84.56	77.41	63.58
2048	89.42	89.31	84.53	77.33	64.62
8192	–	88.85	84.90	77.41	58.53
16384	–	–	84.81	77.26	73.40
32768	–	–	–	78.03	73.40
Full	90.31	89.45	85.03	78.87	75.17

Top-k Performance on AlpacaEval Figure 5 shows that the *95% performance with 2% of attention* result holds on generation-intensive tasks as measured by AlpacaEval 2.0. The trend of small k values achieving near-baseline performance holds true across model sizes, and additional results showing performance across model generations can be found in the appendix (Figure 13).

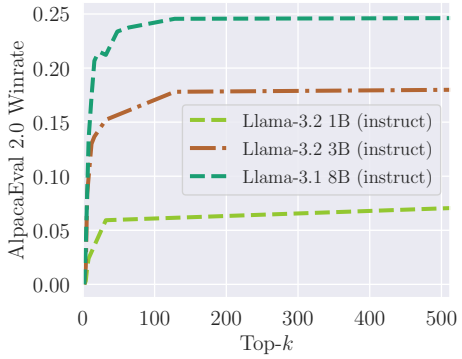


Figure 5: k equivalent to 2% of context length is sufficient to achieve 95% of dense-attention performance on AlpacaEval 2.0, regardless of model size.

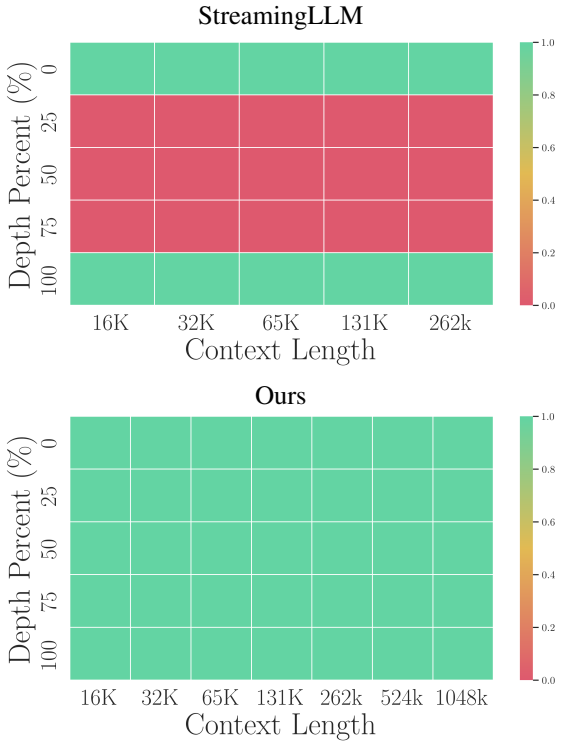


Figure 6: One million token NIAH performance comparing cache eviction (Xiao et al., 2023) and top-k attention. The red cells show that attention sinks are incapable of retrieving tokens outside of the local window or early sink tokens. Top-k attention achieves 100% success with just $k = 10$.

4.3. 1M Token Generation with Top-k

To demonstrate the extreme scaling that top-k attention permits, we use our method to generate tokens conditioned on a context window of *one million* tokens. We choose RULER's Needle In A Haystack task to use for the context, and we use GradientAI's Llama-3-8B model that was trained out to a context length of 1M (Gradient Team, 2024). We run this experiment on a single GPU with a Faiss vector database prefilled with the contents of a KV cache.[1] We also compare our method against (Xiao et al., 2023) for needle in a haystack in Figure 6. Note that top-k attention has a 100% success rate regardless of where in the context the needle is hidden, which cannot be said for any available cache-eviction method.

4.4. Exploring Non-Uniform Choices of k

In this final section we report the results of our studies on the minimal k required for strong performance across different task types and on the impact of applying our top-k operator non-uniformly over the layers of the transformer.

Optimal k Across Task Types Table 2 shows how the value of k necessary to achieve 95% of base model performance varies across the tasks in RULER. For the needle in a haystack task, only a tiny fraction of the entire context is necessary to achieve 95% of the base performance of the model, whereas nearly 9% is necessary in the case of Word Counting. Most of the tasks fall under the 1% line.

Table 2: Percent of attention scores required to reach 95% of dense attention performance for different categories of tasks from the RULER benchmark. k requirements were measured by performance on contexts from 8,000-128,000 tokens. All samples from Llama-3-8B.

Task Category	k Required For 95% Performance
Needle In A Haystack	0.001%
Variable Tracking	0.11%
Question Answering	0.23%
Multiple NIAH	0.27%
Word Counting	8.87%

Layer-Wise Settings for k Recent work has shown that later layers of a transformer network tend to be less crucial to the computation than earlier ones (Gromov et al., 2024). Our method naturally admits a dimension of flexibility as to where efficiency gains are extracted. As some layers may need a larger k budget than others to accurately capture their attention distributions, we explore allowing the value of k to vary across layers.

In Figure 7 we vary k and plot the RULER scores as before, but we allocate our budget for k across layers in two different ways. In the uniform strategy, each layer gets the same value of k. In the adaptive strategy, we linearly increase k from the first to the last layer. In each experiment (curve on chart), the sum of k over all layers is equivalent for both strategies. We find we are able to gain non-trivial performance boosts simply by changing our strategy for how set k for each layer given a fixed computational budget.

5. Related Work

The problem of making transformer model inference more efficient has been studied from many angles. We briefly survey some of the relevant work below.

5.1. Systems Approaches

There is a long line of work in systems solutions to scale to long contexts in LLMs at inference. Flash Attention, and later Flash-Attention 2, provide theoretical linear memory complexity over the sequence length (Dao et al., 2022; Dao,

[1] We use a variety of k values for this experiment find that k =1 is sufficient to solve Needle In A Haystack with a context length of 1 million tokens.

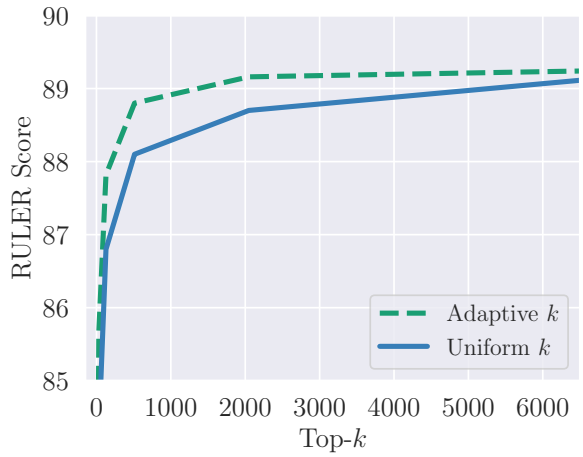


Figure 7: RULER performance of top-k with a fixed vs adaptive k budget across layers. The x axis represents the total k budget, and lines are given for two k budgeting schemes: equal k across all layers and a linearly increasing k from the first to the last layer.

2024). vLLM's implementation of Paged Attention (Kwon et al., 2023) optimizes for throughput over many requests, and, like Flash Attention, this Paged Attention implements a block matrix multiplication algorithm that allows for memory savings at inference time. vLLM, while very performant, does not perform any offloading and assumes the cache will be able to fit on the GPU. Liu et al. (2023) propose Ring Attention, which employs blockwise computation for the self-attention and feedforward operations, distributing long sequences over multiple devices and overlapping key-value block communication with the computation of attention. While the method is highly scalable it assumes access to datacenter-level compute. To the best of our knowledge our approach is the first to achieve inference on million token context windows on a single commodity GPU.

5.2. Cache Eviction Methods

A variety of methods exist for evicting tokens from the cache in order to save GPU memory. A straightforward solution is sliding window attention (Beltagy et al., 2020b), which keeps only recent tokens of a fixed window size in the cache, and assumes that local interactions dominate. StreamingLLM (Xiao et al., 2024) discovered the "attention-sink" phenomenon and proposed a modified sliding window attention that alleviates the performance degradation in windowed attention. Both these works can fail on long contexts as important tokens in the middle of the context window may become evicted from the cache. Another widely used method is H20 (Zhang et al., 2023), a cache eviction strategy that uses information about past tokens to remove tokens from the cache. While effective in increasing decode speeds, this method is not dynamically adjustable to different queries, leading some tokens to be evicted that may be important later. Our method keeps all tokens in the cache offloaded to cheap and plentiful CPU memory, but accelerates the lookup process with fast approximate nearest neighbors.

5.3. Top-k and Dynamic Algorithms

The closest approaches to our method in the literature are all based around the initial work on top-k in Gupta et al. (2021). This method computes all scores directly before filtering out the top-k, incurring quadratic computation and demanding the entire KV cache fits on GPU memory. Chen et al. (2024) also accelerate decoding by selecting relevant tokens, but take an approach based on sampling attention distributions. Singhania et al. (2024) construct a low-cost proxy to full attention using PCA, which informs the token subset for the attention computation. However, these solutions do not support context lengths past a few hundred thousand.

Tang et al. (2024) dynamically select relevant tokens from the cache during decoding, but their method relies on a heuristic approximation of top-k divided over cache pages, whereas our method utilizes a nearest-neighbor data structure. The concurrent works of (Liu et al., 2024a) and (Zhang et al., 2024) bear a resemblance to our method but enforce exactly which kind of nearest-neighbor search algorithm is employed: a custom database in the first and the method of PQ quantization in the second. Our method both generalizes their method to allow for any database and is extended to 1 million length contexts and beyond.

6. Conclusion

In this work we demonstrate the capability of a top-k attention mechanism to operate at the million token scale on a single GPU. We achieve sublinear complexity and evaluate at over 95% of dense attention accuracy on common benchmarks while using only 2% of the context length on average in the attention block. This exploitation of attention sparsity opens up new directions for efficient and viable solutions to long context inference in language models. Our investigation of attention distributions across layers points to future variations of our method that smoothly adapt the top-k method across tasks and layers of a given model suggesting our top-k approach can be used to achieve optimal compute-performance tradeoffs given a specific deployment context.

References

AI, M. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/, 2024.

Accessed: 2024-10-01.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv:2004.05150*, 2020a.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020b.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Chen, Z., Sadhukhan, R., Ye, Z., Zhou, Y., Zhang, J., Nolte, N., Tian, Y., Douze, M., Bottou, L., Jia, Z., and Chen, B. Magicpig: Lsh sampling for efficient llm generation, 2024. URL https://arxiv.org/abs/2410.16179.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers, 2019. URL https://arxiv.org/abs/1904.10509.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mZn2Xyh9Ec.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Dubois, Y., Galambosi, B., Liang, P., and Hashimoto, T. B. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

Gradient Team. Scaling rotational embeddings for long-context language models. https://gradient.ai/blog/scaling-rotational-embeddings-for-long-context-language-models, May 2024. Accessed: 2024-10-01.

Grattafiori, A. et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Gromov, A., Tirumala, K., Shapourian, H., Glorioso, P., and Roberts, D. A. The unreasonable ineffectiveness of the deeper layers, 2024. URL https://arxiv.org/abs/2403.17887.

Gupta, A., Dar, G., Goodman, S., Ciprut, D., and Berant, J. Memory-efficient Transformers via Top-k Attention. In *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing*, pp. 39–52, Virtual, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.sustainlp-1.5.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2020.

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., and Ginsburg, B. RULER: What's the Real Context Size of Your Long-Context Language Models?, April 2024.

Kamradt, G. Needle in a haystack - pressure testing llms. https://github.com/gkamradt/LLMTest, 2023. GitHub repository.

Keisuke, S., Ronan, L. B., Chandra, B., and Yejin, C. Winogrande: An adversarial winograd schema challenge at scale. 2019.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Liu, D., Chen, M., Lu, B., Jiang, H., Han, Z., Zhang, Q., Chen, Q., Zhang, C., Ding, B., Zhang, K., Chen, C., Yang, F., Yang, Y., and Qiu, L. Retrievalattention: Accelerating long-context llm inference via vector retrieval, 2024a. URL https://arxiv.org/abs/2409.10516.

Liu, H., Zaharia, M., and Abbeel, P. Ring attention with blockwise transformers for near-infinite context, 2023. URL https://arxiv.org/abs/2310.01889.

Liu, H., Yan, W., Zaharia, M., and Abbeel, P. World model on million-length video and language with blockwise ringattention, 2024b. URL https://arxiv.org/abs/2402.08268.

Malkov, Y. A. and Yashunin, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836, April 2020. ISSN 0162-8828. doi: 10.1109/TPAMI.2018.2889473. URL https://doi.org/10.1109/TPAMI.2018.2889473.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text, 2016. URL https://arxiv.org/abs/1606.05250.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: high-throughput generative inference of large language models with a single gpu. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Singhania, P., Singh, S., He, S., Feizi, S., and Bhatele, A. Loki: Low-rank keys for efficient sparse attention, 2024. URL https://arxiv.org/abs/2406.02542.

Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B., and Han, S. Quest: Query-aware sparsity for efficient long-context llm inference, 2024. URL https://arxiv.org/abs/2406.10774.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *arXiv*, 2023.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. In *ICLR*. OpenReview.net, 2024.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL https://arxiv.org/abs/1809.09600.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472.

Zhang, H., Ji, X., Chen, Y., Fu, F., Miao, X., Nie, X., Chen, W., and Cui, B. Pqcache: Product quantization-based kvcache for long context llm inference, 2024. URL https://arxiv.org/abs/2407.12820.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., Wang, Z., and Chen, B. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36: 46595–46623, 2023.

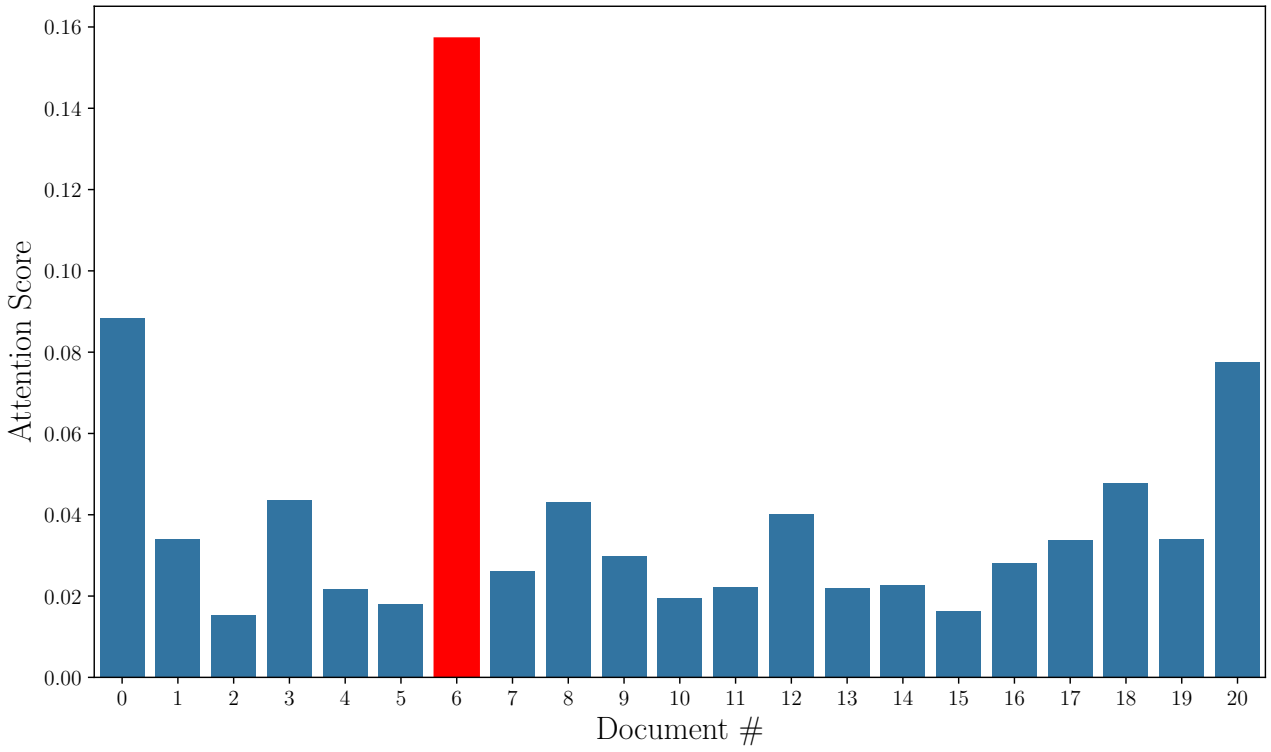


Figure 8: Average attention score per token in a given Wikipedia article for a multi-article long context sample (across all heads and layers). The bar for the article prompted to be copied is highlighted in red. Note how the model focuses its attention on the correct document.

A. Appendix

A.1. Attention Sparsity In The Wild

Next, we take these multi-article samples and prompt the model to copy one of the articles that relates to a specific topic. Figure 8 demonstrates that in expectation the attention scores (across all heads and layers) are higher for the tokens contained within the correct document.

Having confirmed that attention scores are indeed concentrated in terms of both how many tokens are relevant, and which tokens are highlighted in an input sequence, we zoom out and probe for structure in the attention scores in aggregate by quantifying their overall entropy. In Figure 9 we visualize the entropy of the last-token attention scores in a sequence as a function of layer depth. We observe that the entropy is low in all layers and decreases significantly after the first layer. Further analysis of attention sparsity across task categories and intuition on the connection between sparsity and entropy is provided in Appendix A.2.

A.2. Distribution of Attention Scores

In general, 1% of the total attention scores are sufficient for providing 95% of the performance of dense attention on Open LLM Leaderboard, AlpacaEval, and RULER. However, within the subtasks for a given benchmark there is variation in this `k`-required threshold. This variation is highly correlated with the a measurement we call the *attention entropy*. Attention entropy is calculated by taking the Shannon entropy of a single row of an attention matrix after the softmax transformation is applied, and averaging that over multiple tokens of generation on many different samples of text.

When treating a single, soft-maxed row of an attention matrix as a probability distribution, entropy serves as a good descriptor of how concentrated, or ”sparse” it is. The entropy of a maximally concentrated attention distribution is zero, while completely uniform attention scores would have an entropy of the logarithm total scores. Thus low entropy indicates

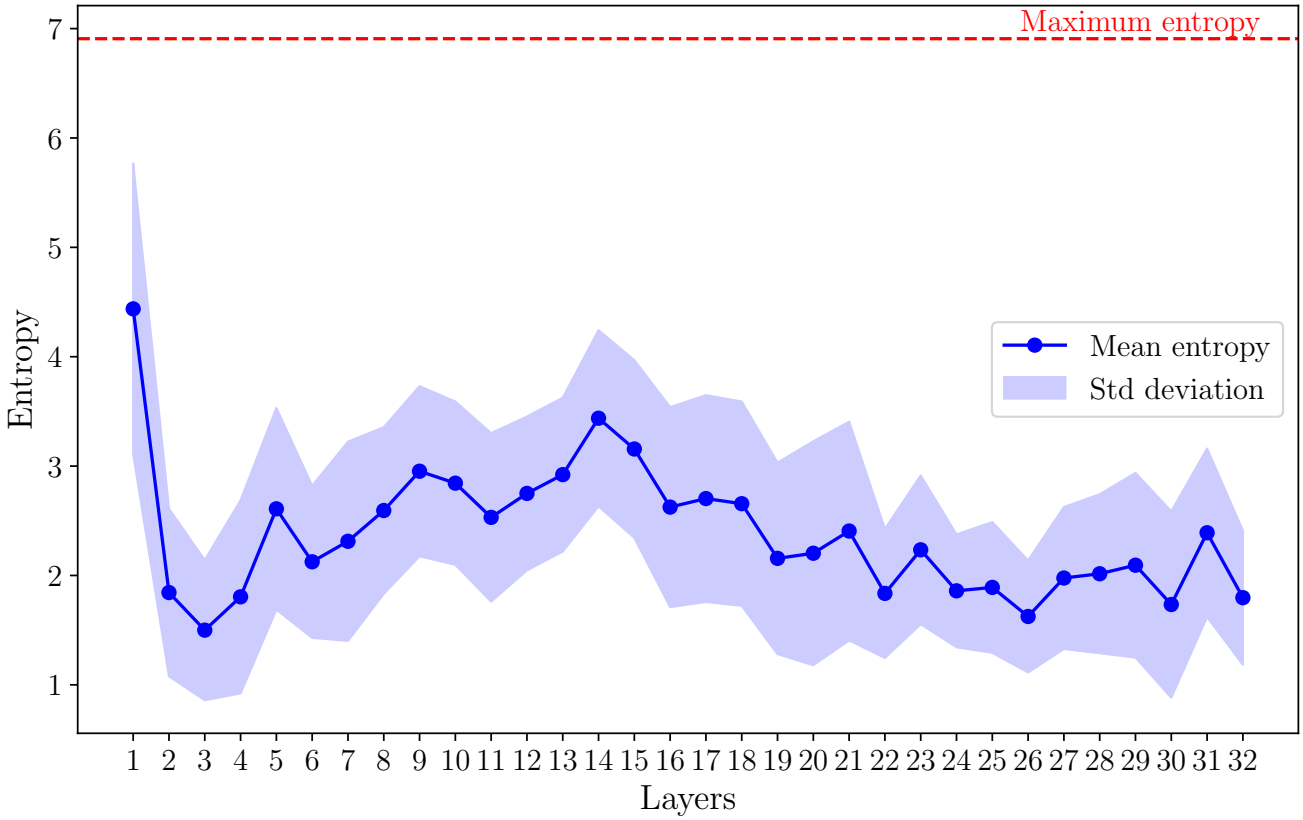


Figure 9: Entropy of the distribution of attention scores for each layer of a model, calculated as $E = -\sum_{i=1}^{N} a_i \log(a_i)$, where $(a_1, \ldots, a_N)$ is the attention score distribution. Attention score distributions are derived from last token of 50 1000-token samples and aggregated over all heads for a given layer. Entropy serves as a measure of how concentrated the attention scores are for a given query token: low entropy indicates a large amount of attention centered over few tokens, and high entropy indicates a more uniform dispersion of attention scores. Maximum entropy occurs when the distribution is perfectly uniform, and for 1000-token contexts is $-\log(\frac{1}{1000})$.

sparse, or concentrated attention. In Table 3 we show the attention entropy calculated from the first ten tokens of generation from fifty samples of text in each task category. The attention entropy values in the table have a Pearson correlation coefficient of 0.85 with the k-required thresholds for 95% performance in those tasks.

In addition to looking at the attention distributions across tasks, we investigate if there are any systematic trends in the attention sparsity across layers of a model. Figure 10 shows the attention entropy for RULER subtasks when plotted by layer, and Figure 3 shows the number of scores required to cover 75% of the full attention distribution, with a histogram plotting this value over hundreds of samples of Wikipedia text. Note that in both figures, the first layer clearly stands out as having the least concentrated attention distributions.

A.3. Additional RULER and AlpacaEval Results

Table 3: Percent of attention scores required to reach 95% of dense attention performance for different categories of tasks, along with the average entropy of the attention vectors for tokens generated from those tasks.

Task Category	**k Required For 95% Performance**	**Attention Entropy**
Needle In A Haystack	0.001%	1.93
Variable Tracking	0.11%	2.11
Question Answering	0.23%	2.27
Multiple NIAH	0.27%	2.33
Word Counting	8.87%	2.68
k-Required - Entropy Correlation (r)	0.847	

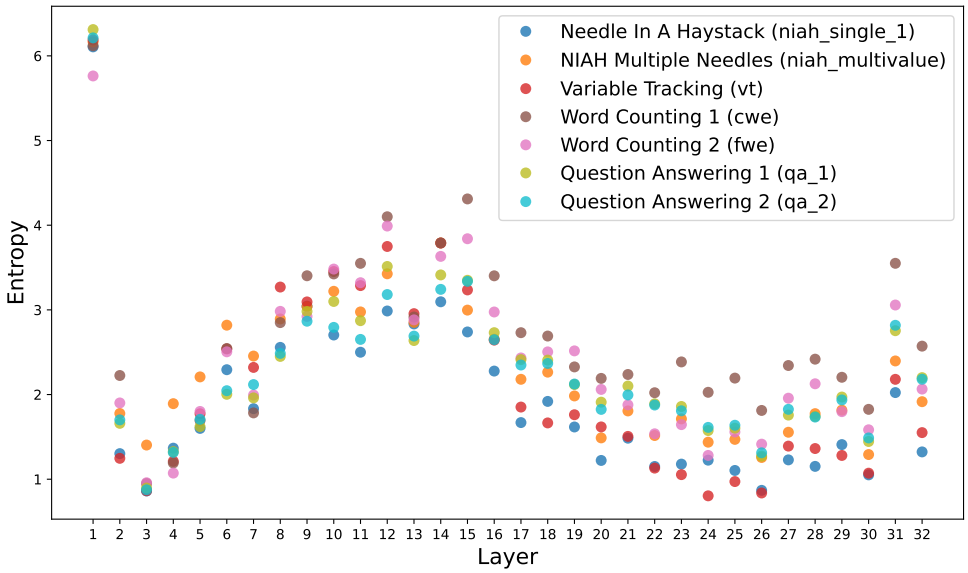


Figure 10: Attention entropy by layer, colored by task category.

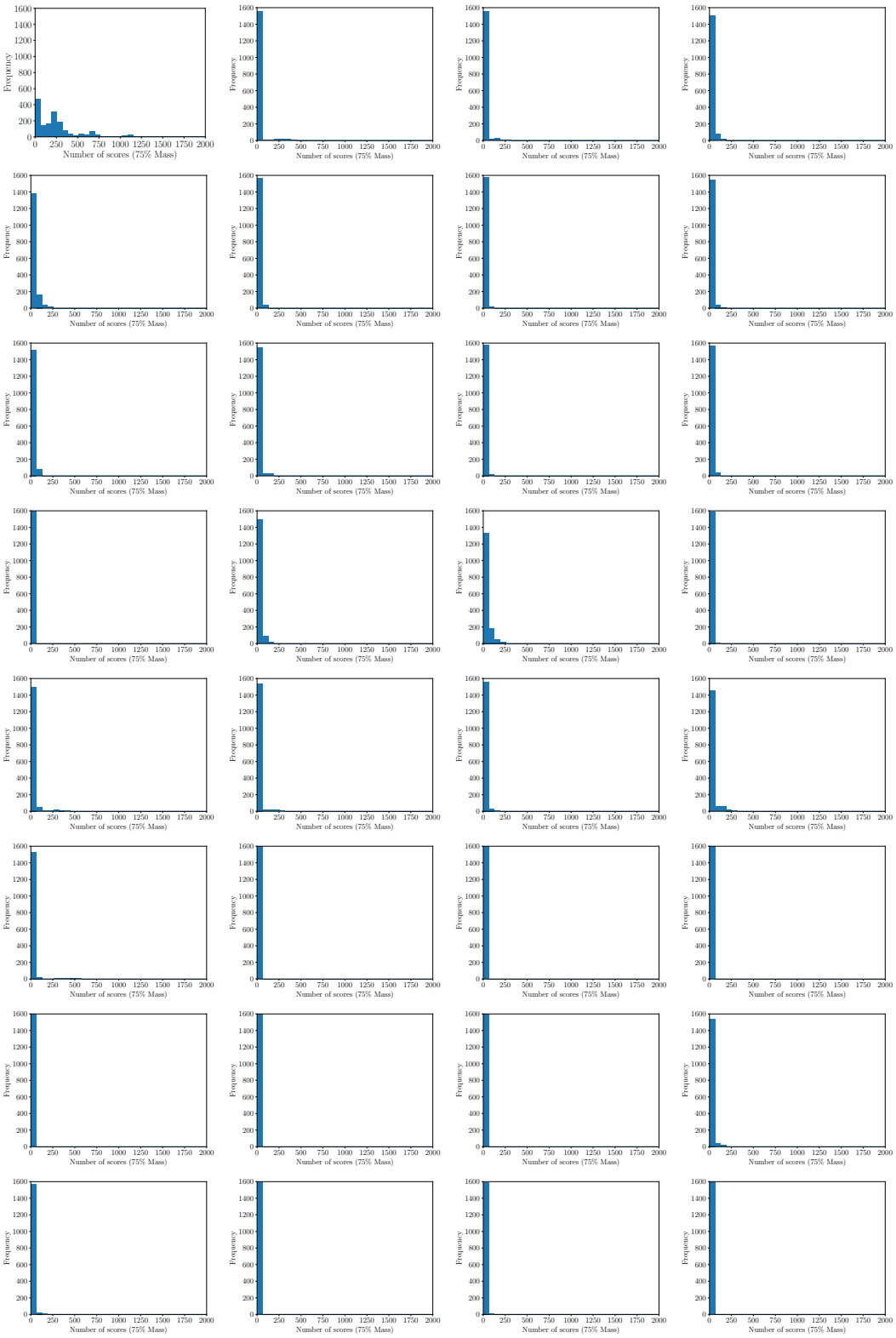
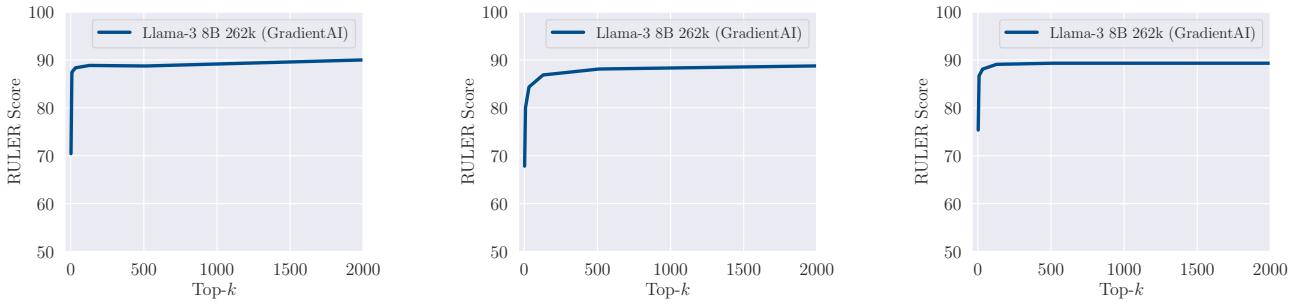
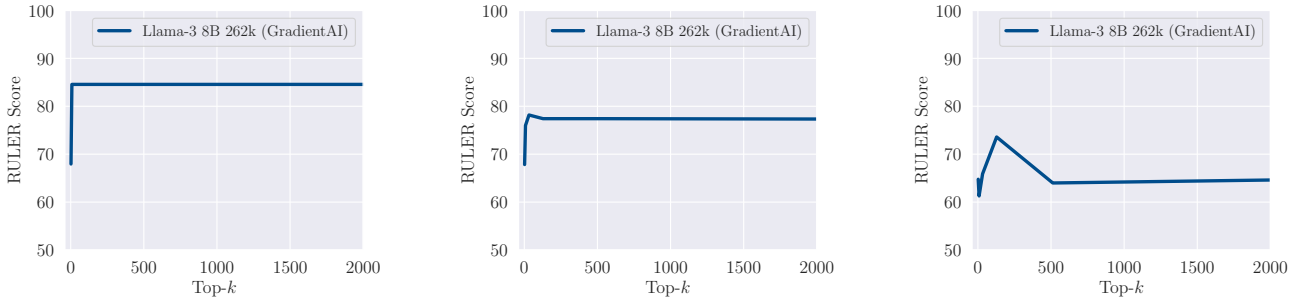


Figure 11: All 32 layers are plotted in order, where the top row represents layers 1, 2, 3, and 4 and last row represents layers 29, 30, 31, and 32.



(a) Context length 4096 (b) Context length 8192 (c) Context length 16384



(d) Context length 32768 (e) Context length 65536 (f) Context length 131072

Figure 12: Results for RULER over various context lengths. This shows the same behavior as Open LLM Leaderboard and AlpacaEval where very few attention scores (very low k) are sufficient to achieve near dense attention performance.

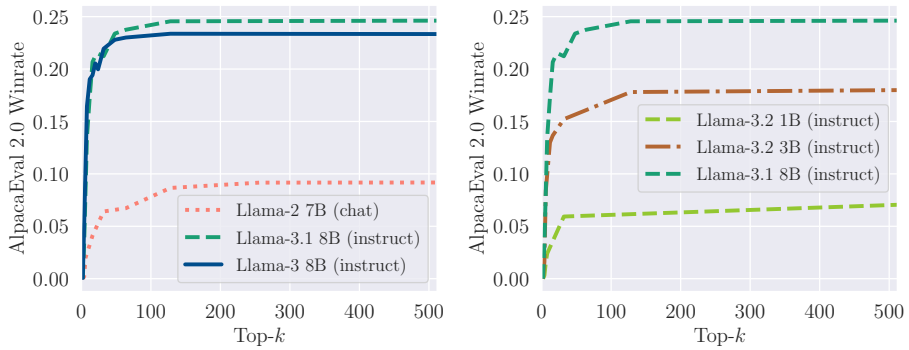


Figure 13: AlpacaEval 2.0 results for various models. Left compares different generations of Llama instruction tuned models. Right investigates how models of different sizes handle small values of k.