arXiv:2312.04985v6 [cs.LG] 4 Sep 2024

SparQ Attention: Bandwidth-Efficient LLM Inference

Luka Ribar [* 1] Ivan Chelombiev [* 2] Luke Hudlass-Galley [* 1] Charlie Blake [1] Carlo Luschi [1] Douglas Orr [1]

Abstract

The computational difficulties of large language model (LLM) inference remain a significant obstacle to their widespread deployment. The need for many applications to support long input sequences and process them in large batches typically causes token-generation to be bottlenecked by data transfer. For this reason, we introduce **SparQ Attention**, a technique for increasing the inference throughput of LLMs by utilising memory bandwidth more efficiently within the attention layers, through selective fetching of the cached history. Our proposed technique can be applied directly to off-the-shelf LLMs during inference, without requiring any modification to the pre-training setup or additional fine-tuning. We show that SparQ Attention brings up to $8\times$ savings in attention data transfers without substantial drops in accuracy, by evaluating Llama 2 and 3, Mistral, Gemma and Pythia models on a wide range of downstream tasks.

1. Introduction

Transformer models trained on large corpora of text have recently shown remarkable performance on complex natural language processing tasks (Achiam et al., 2023; Touvron et al., 2023). This has been attributed to the *in-context learning* capabilities that emerge with large-scale training, enabling arbitrary textual information (e.g. long instructions, chat histories, relevant documents) to be incorporated at inference-time (Wei et al., 2022).

To leverage the benefits of in-context learning, there has been demand for LLMs to support increasingly long input sequences. However, the standard inference optimisation

*Equal contribution [1]Graphcore Research, United Kingdom [2]Synthesia, United Kingdom (work done while at Graphcore Research). Correspondence to: Luka Ribar, Luke Hudlass-Galley, Douglas Orr {lukar, lukehg, douglaso}@graphcore.ai.
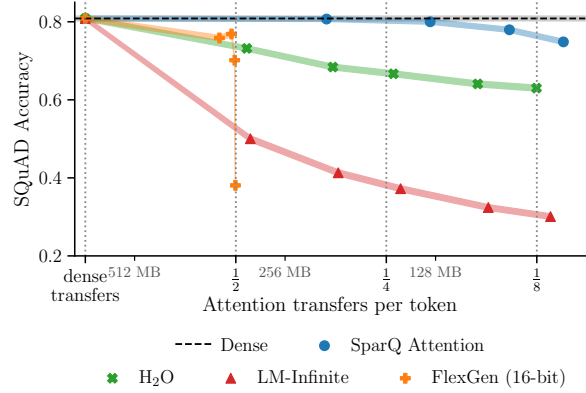
Figure 1: Llama 2 13B SQuAD 1-shot performance versus attention transfers over a range of compression ratios. SparQ Attention achieves matching performance, while transferring between $1/8$ and $1/4$ as much data as the original dense model. Line thickness shows $\pm$ one standard error over 4000 examples (the uncertainty from a finite test set). This pattern is representative of the performance across various models and tasks, shown in Figures A1 to A3.

used to support in-context learning, *key-value (KV) caching* (Pope et al., 2023), is constrained by the need to fetch a large amount of data from memory when processing batches of long sequences. This in turn limits the speed at which tokens can be generated—a key usability metric for LLMs.

This bottleneck can be attributed to the auto-regressive nature of transformer generation. For each token generated, the full KV cache must be fetched from memory. The size of the KV cache scales linearly with the sequence length, as well as the batch size, thus rendering generation for long batched sequences increasingly memory bandwidth limited.

Despite this expensive cache-fetch at each step, tokens generally only attend to a small part of the sequence at a time (Vig, 2019; Yun et al., 2020). If it were possible to efficiently predict which tokens will have high attention scores, memory bandwidth efficiency could be significantly increased by only transferring the key-value pairs of high-scoring tokens.

Building upon this idea, we present **SparQ Attention**, a technique for significantly improving the memory bandwidth efficiency of transformer inference. By approximating attention scores using a subset of query and key components,
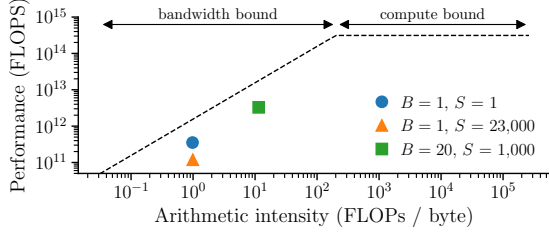
Figure 2: Roofline analysis of Llama 2 7B on A100 (40GB), highlighting that for a range of LLM inference settings with batch size B and sequence length S, practical performance is memory bandwidth bound.
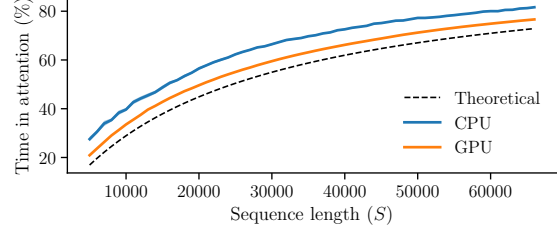

Figure 3: The proportion of time that is spent in attention layers during Llama 2 7B inference with a single sample when using `llama.cpp` on both CPU and GPU platforms. For more details, see Appendix D.

we fetch only the most relevant tokens for each generation step, reducing the amount of data transferred without degrading the model.

We also provide a new set of challenging downstream task variants which we use to evaluate SparQ Attention. These are based on existing tasks, modified to assess a model's ability to utilise information from long input sequences for multi-token generation. We show that SparQ Attention performs favourably compared to other state-of-the-art methods, giving up to $8\times$ compression without substantial loss in accuracy. SparQ Attention is robust across tasks and models, demonstrated by evaluation on Llama 2 and 3, Mistral, Gemma and Pythia. We also provide benchmarks measured on IPU and GPU, showing the practical computational benefits of our approach.

2. Background

In this section we provide a straightforward framework to understand the computational efficiency of sequence generation using transformer models (similar to the modelling introduced by Kaplan et al. (2020)) and use it to motivate transfer-efficient attention mechanisms.

Arithmetic intensity Consider a compute unit capable of $r_{\mathcal{A}}$ scalar arithmetic operations per second that is connected to a memory via an interface which can transfer $r_{\mathcal{M}}$ scalar elements per second. Given a workload requiring $\mathcal{A}$ arithmetic operations and $\mathcal{M}$ transfers, and assuming concurrent compute and data transfer, the *arithmetic intensity* is defined as $\mathcal{A}/\mathcal{M}$. In LLM inference, $\mathcal{A}$ is primarily a function of the size of matrix multiplications in the model, and $\mathcal{M}$ depends on various factors such as the size of the KV cache, model size, and batch size. When the arithmetic intensity of the workload is less than the ratio $r_{\mathcal{A}}/r_{\mathcal{M}}$, execution time is limited by $r_{\mathcal{M}}$, due to the data transfer taking longer than the computation in the concurrent setting.

The arithmetic intensity of typical sequence generation workloads in transformer models is shown in Figure 2, high-

lighting that execution time is bandwidth bound, not compute bound. We provide a more general analysis of the arithmetic intensity of sequence generation in Appendix C, showing that it is typically bandwidth bound. A corollary of this is that the most effective way to accelerate transformer sequence generation is to reduce data transfers.

Time in attention Sequence generation with transformers is dominated by two types of computation. The first is a position-wise matrix multiplication between activations and parameters. The second is dot-product self-attention between activations (Vaswani et al., 2017). Assuming a standard transformer layer with model dimension d_m, batch size B, sequence length S and using Grouped Query Attention (GQA) (Ainslie et al., 2023) with g grouped-query heads per key-value head ($g=1$ for standard multi-head attention), the proportion of data transfers associated with attention is given by

$$\frac{\mathcal{M}_{\text{attn}}}{\mathcal{M}_{\text{attn}} + \mathcal{M}_{\text{params}}} = \frac{\rho}{\rho + 6/B}, \quad (1)$$

where $\mathcal{M}_{\text{params}}$ and $\mathcal{M}_{\text{attn}}$ are the total data transfers of parameters and KV cache respectively, and $\rho = S/(gd_m)$ is a variable we have introduced to capture the relevant model hyperparameters (see Appendix C). When $\rho \gg 6/B$ (for example, with large S or B), attention dominates data transfers, as the entire KV cache must be transferred during each generative step. This theoretical trend is backed up by empirical results from `llama.cpp` (Gerganov, 2024) benchmarks in Figure 3.

Since data transfer is the performance-limiting factor, and attention transfers dominate as sequence length is increased, there is a need for transfer-efficient alternatives to the standard attention mechanism.

3. Approximating Attention

In this section we examine several properties of the attention operation that enable us to introduce an accurate bandwidth-efficient approximation.
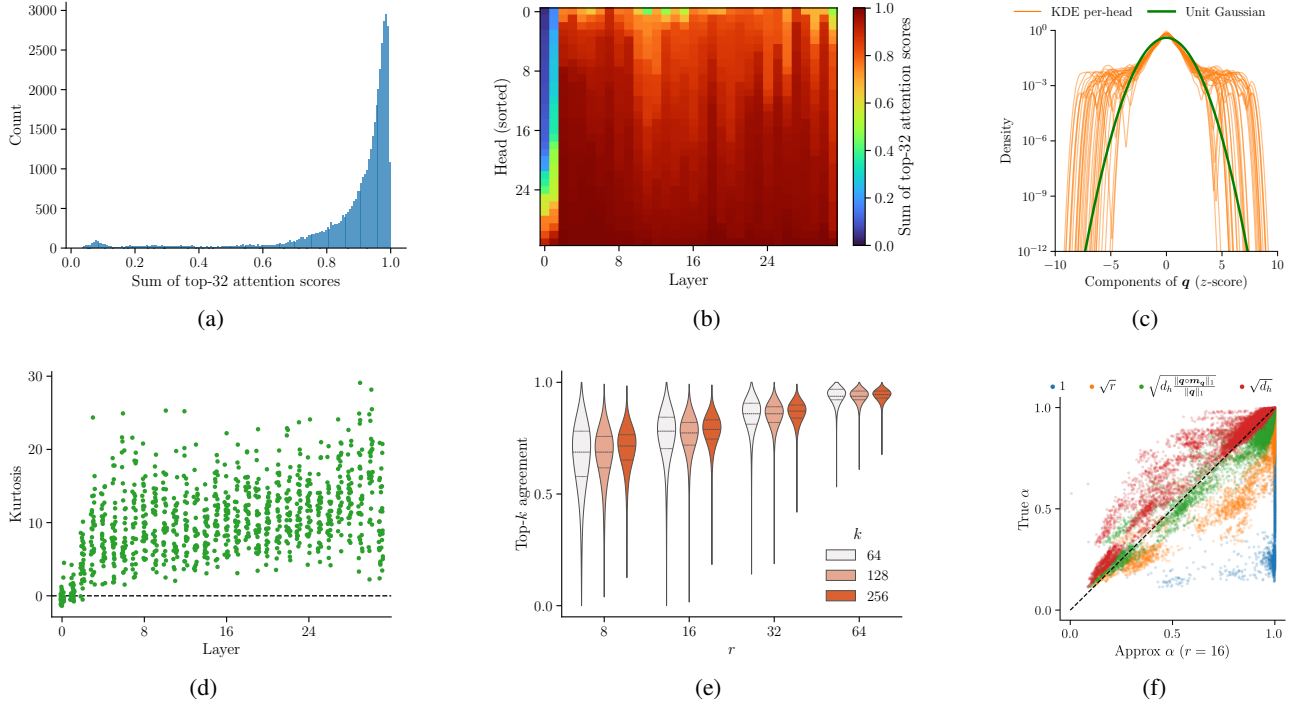
Figure 4: Statistics of Llama 2 7B, evaluated over 40 SQuAD queries, over all 32 layers × 32 heads unless noted. (a) Sum softmax output allocated to the 32 highest-scoring positions, demonstrating natural attention sparsity; (b) for each head. (c) Kernel density estimate (Rosenblatt, 1956) of components of q in layer 16, showing heavy tails. (d) Fisher Kurtosis of q components, for each head, showing that the query vector is leptokurtic for most heads. (e) Top-k agreement between approximate and true scores for multiple values of r selected from query vector. Top-k agreement is the proportion of the top-k positions that are correctly predicted by an approximated softmax, using a projection of q. (f) Agreement between the coverage α based on estimated scores versus the true mass of the top 128 scores, for different softmax temperatures (a point for each example × head), showing the importance of correct temperature. Further analysis is presented in Appendix E.

Consider a single attention *query head* with the head dimension d_h, processing an input token sequence of length S. During autoregressive generation, the output of the attention head is calculated as:

$$y = \text{softmax}\left(\frac{q \cdot K^\top}{\sqrt{d_h}}\right) \cdot V \qquad (2)$$

where q is the query, and $K \in \mathbb{R}^{S \times d_h}$ and $V \in \mathbb{R}^{S \times d_h}$ are the key and value caches respectively. When using GQA (Ainslie et al., 2023), K and V are shared across g query heads.

For each forward pass, we need to fetch the key and value matrices from memory, as well as write (append) k and v vectors for the current token, giving a total number of elements transferred per attention head:

$$\mathcal{M}_\text{dense} = 2 S d_h + 2 d_h \qquad (3)$$

where the first term corresponds to reading the K and V caches and the second term corresponds to writing the cur-

rent k and v to memory. Memory transfer may be equivalently expressed in terms of bytes, however we use scalar elements to disentangle cache compression methods from the number format used to represent the cache.

Attention scores sparsity   First, consider the *attention scores* $s \in (0, 1)^S$ in Equation (2):

$$s = \text{softmax}\left(\frac{q \cdot K^\top}{\sqrt{d_h}}\right) \qquad (4)$$

Due to the normalising effect of the softmax function, the resulting s vector is *sparse* (see Figures 4a and 4b), i.e. we can find a boolean mask $m_s \in \{0, 1\}^S$ corresponding to the top-k elements in s ($k \ll S$) such that:

$$y_1 = (s \circ m_s) \cdot V \approx s \cdot V \qquad (5)$$

As a result, only the values v_i corresponding to the non-zero elements of m_s need to be fetched from memory. However, the algorithm still requires fetching the full K from memory in order to calculate the attention scores s, limiting the minimum amount of data transferred to $\frac{1}{2}\mathcal{M}_\text{dense}$.

Table 1: *Excess* correlation ratio η (Roche et al., 1998) along axes of V (excess: subtract $d^{-0.5}$, so uniform random data = 0.0). This demonstrates substantial auto-correlation along the sequence axis. Calculated for Llama 2 7B over 40 SQuAD examples.

	B	S	Layer	Head	d_h
$\eta - d^{-0.5}$	0.143	**0.256**	0.0	0.0	0.0

Mean value reallocation In order to further improve the approximation in Equation (5), we note a further observation: v_i vectors within the sequence exhibit a high degree of auto-correlation (see Table 1). Thus, an additional correction term using a running-mean value vector $\bar{v} = \frac{1}{S} \sum_{i=1}^{S} v_i$ can be added as follows:

$$y_2 = (s \circ m_s) \cdot V + (1 - s \cdot m_s)\bar{v} \qquad (6)$$

This introduces a minimal additional overhead compared to Equation (5) due to the mean vector $\bar{v}$ being updated and written back to memory at each step.

Query sparsity In order to improve the lower bound on memory transfers, we further consider efficiently approximating the mask m_s by calculating approximate attention scores $\hat{s}$ without using the full matrix K. Here, we consider the distribution of magnitudes of the components of the query vector q and observe that it is highly *heavy-tailed* (see Figures 4c and 4d). This observation allows us to efficiently approximate the attention scores s by defining a per-query boolean mask $m_q \in \{0,1\}^{d_h}$ corresponding to the top-r components of q. The scores are then approximated as:

$$\hat{s} = \text{softmax}\left(\frac{(q \circ m_q) \cdot K^\top}{\tau}\right) \qquad (7)$$

where τ is the softmax temperature. Due to the mask m_q, only the components of K corresponding to non-zero elements of the mask need to be fetched from memory. The top-k mask $m_{\hat{s}} \in \{0,1\}^S$ can then be calculated using $\hat{s}$ (see Figure 4e) and the approximate attention output is obtained as:

$$y_3 = \text{softmax}\left(\frac{q \cdot K^\top}{\sqrt{d_h}} + \log(m_{\hat{s}} + \epsilon)\right) \cdot V \qquad (8)$$

with $\epsilon \to 0$. Again, due to the mask $m_{\hat{s}}$, only the key-value pairs corresponding to the non-masked elements need to be fetched from the memory.

Mean value reallocation with query sparsity As a final consideration, we look at combining the mean value reallocation improvement of Equation (6) with the approach in Equation (8). As we do not have access to the full scores

s, we proceed to approximate the weighted sum using the approximate scores in Equation (7). Note that, since the query-key dot product is performed over only r dimensions, care needs to be taken when choosing the appropriate softmax temperature τ in Equation (7). If r components were chosen randomly, the appropriate temperature would be $\sqrt{r}$. On the other hand, if the top-r components were the only non-zero elements of the query vector, the appropriate temperature would remain $\sqrt{d_h}$. As a balance between the two extremes, we have found the following temperature to yield a good approximation (see Figure 4f):

$$\tau = \sqrt{d_h \frac{\|q \circ m_q\|_1}{\|q\|_1}} \qquad (9)$$

The final attention output can be then calculated as a weighted sum:

$$y = \alpha \, y_3 + (1 - \alpha)\bar{v} \qquad (10)$$

where $\alpha = m_{\hat{s}} \cdot \hat{s}$ is the relative weight of the top-k terms.

4. SparQ Attention

Following the analysis in Section 3, we propose **SparQ Attention** (see Figure 5) consisting of three steps:

Step 1: Find the indices of r largest components of $|q|$[1] and only fetch K along the dimensions corresponding to these indices. Calculate *approximate* attention scores $\hat{s}$ using the sliced query and keys.

Step 2: Find the top-k positions in the approximate attention scores and fetch the corresponding *full* key and value vectors. Calculate the output of the attention operation using the top-k keys and values.

Step 3: Estimate the total score α assigned to the top-k positions using the approximate attention scores. Use this total score to interpolate between the attention output from the top-k positions, and a *mean value* vector, $\bar{v}$.

The memory transfer of the SparQ Attention algorithm for a single attention head forward-pass:

$$\mathcal{M}_{\text{SparQ}} = S\,r + 2\,k\,d_h + 4\,d_h \qquad (11)$$

where the first term corresponds to reading r rows of K, the second term corresponds to reading the top-k columns of K and V and the third term corresponds to transfers associated with writing the current k and v, in addition to reading and writing $\bar{v}$.

---

[1]We use $|\cdot|$ to denote element-wise absolute value.
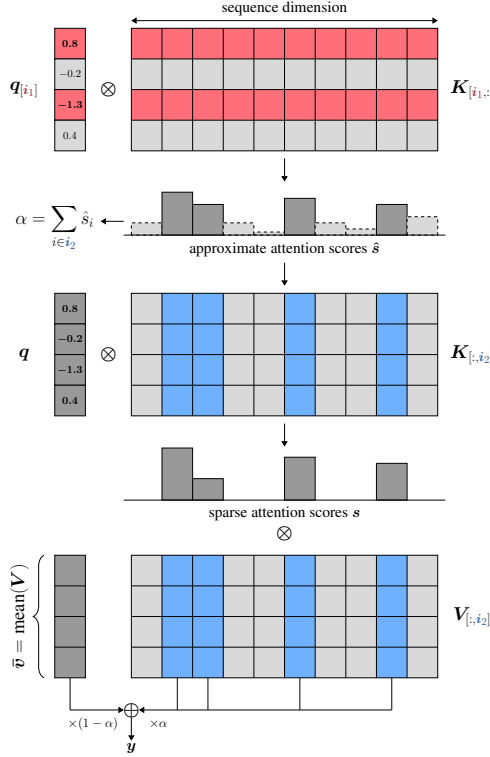
Algorithm 1 SparQ Attention

Input: $\boldsymbol{q} \in \mathbb{R}^{d_h}$, $\boldsymbol{K} \in \mathbb{R}^{S \times d_h}$, $\boldsymbol{V} \in \mathbb{R}^{S \times d_h}$, $\overline{\boldsymbol{v}} \in \mathbb{R}^{d_h}$, $r \in \mathbb{N}, k \in \mathbb{N}, l \in \mathbb{N}$

 # Indices of top r elements of $|\boldsymbol{q}|$
$\boldsymbol{i}_1 \leftarrow \mathrm{argtopk}\left(|\boldsymbol{q}|, r\right)$

 # Softmax temperature, weighted by L1 coverage
$$\tau \leftarrow \sqrt{d_h \cdot \frac{\|\boldsymbol{q}_{[\boldsymbol{i}_1]}\|_1}{\|\boldsymbol{q}\|_1}}$$

 # Approximate attention scores (all positions)
$\hat{\boldsymbol{s}} \leftarrow \mathrm{softmax}\left(\boldsymbol{q}_{[\boldsymbol{i}_1]} \cdot \boldsymbol{K}_{[\boldsymbol{i}_1,:]}^{\top} / \tau\right)$

 # Local mask of last l positions
$\boldsymbol{m} \leftarrow [1 \textbf{ if } i > S - l \textbf{ else } 0]_{i=1}^{S}$

 # Indices of top k approximate scores or local
$\boldsymbol{i}_2 \leftarrow \mathrm{argtopk}\left(\hat{\boldsymbol{s}} + \boldsymbol{m}, k\right)$

 # Total approximate score of top k
$\alpha \leftarrow \mathrm{sum}\left(\hat{\boldsymbol{s}}_{[\boldsymbol{i}_2]}\right)$

 # Final attention scores (top k positions)
$\boldsymbol{s} \leftarrow \mathrm{softmax}\left(\boldsymbol{q} \cdot \boldsymbol{K}_{[:,\boldsymbol{i}_2]}^{\top} / \sqrt{d_h}\right)$

 # Mixed scores and values, interpolating with $\overline{\boldsymbol{v}}$
$\boldsymbol{y} \leftarrow \alpha \, \boldsymbol{s} \cdot \boldsymbol{V}_{[:,\boldsymbol{i}_2]} + (1 - \alpha) \, \overline{\boldsymbol{v}}$

return $\boldsymbol{y}$

Figure 5: SparQ Attention for a single attention head. The algorithm consists of three steps. First, we find the r largest components of the incoming query vector and gather the corresponding components along the hidden dimension of the key cache $\boldsymbol{K}$. This allows us to approximate the full attention scores ($\hat{\boldsymbol{s}}$). In the second step, we identify the top-k largest scores in the approximation and proceed to gather the corresponding full key and value vectors from the cache. As a final step, to compensate for the missing value vectors, we additionally maintain and fetch the running mean value vector $\overline{\boldsymbol{v}}$ and reassign it the leftover mass based on approximate score weightings. The attention output is then calculated as usual using the top-k fetched key and value pairs, together with $\overline{\boldsymbol{v}}$.

By varying r and k, we can tune the total amount of data transferred by the scheme, trading-off approximation accuracy for token-generation speed-up. Since typically $S \gg d_h$, r is the most important parameter controlling the data transfer compression ratio $\mathcal{M}_{\mathrm{SparQ}} / \mathcal{M}_{\mathrm{dense}}$.

Grouped query attention   For models using GQA, groups of g queries access the same KV head. In order to accommodate this, we modify **Step 1** to sum $|\boldsymbol{q}|$ within each group before selecting top-r components. Similarly, **Step 2** is modified by summing the approximate attention scores within each group before selecting top-k keys and values for each KV head. Although **Step 3** can be implemented exactly as before, we found that GQA models obtained better performance without it, so we omitted this step for Llama 3 and Mistral. The full code can be found in Appendix B.

5. Experiments

5.1. Setup

Models   We evaluate our method on five widely-used open-source language model variants: Llama 2 (Touvron et al., 2023), Llama 3 (Meta AI, 2024), Mistral (Jiang et al., 2023), Gemma (Mesnard et al., 2024) and Pythia (Biderman et al., 2023), evaluating model sizes up to 13 billion parameters.[2] All models are decoder-only transformers (Radford et al., 2018), pre-trained on causal language modelling. They share similar architectural components such as rotary positional embedding (Su et al., 2021), while also having some notable differences such as different attention mechanisms (multi-head and grouped query attention), layer normalisation implementations, activation functions and execution of modules in parallel.

Tasks   In order to evaluate our method on a spectrum of relevant NLP tasks that present a particular challenge to sparse attention techniques, our evaluation setup consists of various tasks requiring information retrieval and reasoning over long input sequences. This includes question answering, summarisation, perplexity/bits-per-character (BPC), and text repetition. For this, we adapted standard downstream tasks and datasets to generate examples of sequence lengths be-

---

[2]https://github.com/graphcore-research/llm-inference-research/tree/2024-05-sparq

Table 2: Results for the largest model of each family tested are presented below. SQuAD and TriviaQA measure performance in accuracy as a percentage; CNN/DailyMail uses ROUGE-L score; WikiText task measures perplexity in bits per character (BPC); Repetition counts the number of characters before the generation diverges. Values presented are those closest to the target compression ratio for each technique, where bold represents the best score for each setting. Median standard errors across all models and sparsity settings are: SQuAD 0.8, TriviaQA 0.8, CNN/DailyMail 0.4, WikiText 0.01, Repetition 2.

Dataset Name		SQuAD ↑			TriviaQA ↑			CNN/DailyMail ↑			WikiText ↓			Repetition ↑		
Compression		1	1/2	1/8	1	1/2	1/8	1	1/2	1/8	1	1/2	1/8	1	1/2	1/8
Llama 2 13B	LM-∞	**80.8**	50.0	30.1	**78.7**	73.4	68.1	**22.1**	16.8	14.9	**0.61**	0.64	0.71	**229**	76	29
	H_2O	**80.8**	73.2	63.0	**78.7**	78.5	**78.4**	**22.1**	22.2	20.3	**0.61**	**0.61**	0.64	**229**	61	26
	SparQ	**80.8**	**80.7**	**74.9**	**78.7**	**78.8**	78.2	**22.1**	**22.5**	21.6	**0.61**	**0.61**	0.70	**229**	**227**	**190**
Llama 3 8B	LM-∞	**81.2**	66.0	51.8	**83.2**	81.8	80.6	**23.4**	17.1	16.1	**0.56**	0.58	0.64	**213**	102	27
	H_2O	**81.2**	74.5	61.7	**83.2**	**83.2**	82.5	**23.4**	23.3	21.9	**0.56**	**0.56**	0.59	**213**	67	30
	SparQ	**81.2**	**81.2**	**78.3**	**83.2**	83.0	**82.8**	**23.4**	**23.4**	**23.4**	**0.56**	0.57	**0.58**	**213**	**214**	**213**
Mistral 7B	LM-∞	**81.0**	51.0	29.0	**80.9**	75.8	72.6	**23.7**	18.0	16.6	**0.62**	0.65	0.72	**231**	81	20
	H_2O	**81.0**	71.2	56.9	**80.9**	**80.8**	**80.2**	**23.7**	23.5	22.8	**0.62**	0.63	0.66	**231**	38	14
	SparQ	**81.0**	**80.9**	**77.5**	**80.9**	**80.8**	79.0	**23.7**	23.5	**23.0**	**0.62**	0.63	**0.65**	**231**	**209**	**201**
Gemma 7B	LM-∞	**80.4**	64.2	48.7	**82.8**	81.6	80.8	**17.4**	13.1	13.3	**0.59**	0.61	0.68	**245**	101	23
	H_2O	**80.4**	73.7	60.2	**82.8**	**82.9**	82.5	**17.4**	17.4	16.9	**0.59**	**0.59**	0.62	**245**	68	18
	SparQ	**80.4**	**80.3**	**80.3**	**82.8**	82.8	**82.7**	**17.4**	**17.9**	**18.0**	**0.59**	**0.59**	**0.59**	**245**	**240**	**237**
Pythia 6.9B	LM-∞	**57.8**	38.5	17.0	**52.6**	41.6	29.7	**20.2**	14.9	14.0	**0.68**	0.71	0.79	**150**	64	18
	H_2O	**57.8**	52.9	45.5	**52.6**	**52.6**	**52.3**	**20.2**	20.3	18.5	**0.68**	0.69	0.71	**150**	47	17
	SparQ	**57.8**	**58.0**	**57.1**	**52.6**	52.4	51.7	**20.2**	**20.6**	**20.6**	**0.68**	**0.68**	**0.70**	**150**	**151**	**144**

tween 1k and 2k tokens. To define the tasks independently of the selected models, our examples were chosen to have sequence lengths between 4000 and 8000 characters, roughly giving the desired lengths in tokens.

For question answering, we use the SQuAD (Rajpurkar et al., 2016) and TriviaQA (Joshi et al., 2017) datasets in the *open-book* setting. In order to construct the SQuAD examples, we augment the provided context (i.e. the standard SQuAD input sequence required to answer the question) with seven additional "*confusion contexts*" from unrelated questions. This ensures that the examples have a large sequence length, while making the task harder as the model needs to distinguish the relevant information from the context from the unrelated paragraphs. We use SQuAD v1.1, as it does not include unanswerable questions included in SQuAD v2.0, since we aim to measure the model's ability to extract useful information from the KV cache. For both question answering tasks we use exact string match accuracy as the evaluation metric. Summarisation is evaluated on the CNN/DailyMail dataset (See et al., 2017) using the ROUGE-L F-score (Lin, 2004) as the metric. We use the WikiText-103 dataset (Merity et al., 2016) with bits per character (BPC) for evaluating language modelling performance.[3] Finally, we construct an artificial "Text Repetition" task to evaluate the capability of the model to repeat sentences from its context verbatim. Such a task can commonly appear in a dialogue setting where the LLM agent is re-

___
[3]We quote performance for sub-word language modelling in BPC, to account for any differences in vocabulary across models.

quired to retrieve a piece of text from a possibly long context provided, and can be challenging for sparse attention techniques. We construct examples using the Tiny-Shakespeare dataset (Karpathy, 2015) by chunking the text into contexts of the appropriate size, appending them with the prompts containing a subset of the context, and evaluating the output exact character length match with the continuation from the context.

Baselines We consider the cache eviction technique H_2O (Zhang et al., 2023), top-k sparse attention in the form of FlexGen (Sheng et al., 2023), and LM-Infinite, a local windowing scheme with initial-tokens included proposed by Han et al. (2023) as baselines. For each experiment we fix the KV cache transfer budget k idependently of the sequence length. With H_2O, we set the local window size $l = k/4$ (with $3k/4$ heavy hitters), and for LM-Infinite we always include the first 16 positions (with $k - 16$ local positions). Due to the lower bound of FlexGen's compression ratio being $1/2$, we do not report the technique's results in Table 2 and Table 3, but full results can be found in Appendix A. The compression ratio definitions for each of these techniques can be found in Appendix G.

5.2. Results

Our experiments span eight distinct models: Llama 2 with 7 and 13 billion parameters, Llama 3 with 8 billion parameters, Mistral with 7 billion parameters, Gemma with 7 billion parameters, and three Pythia models with 1.4, 2.8 and 6.9
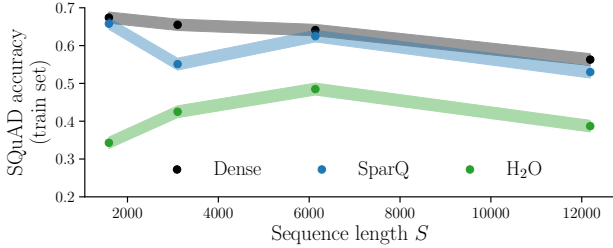
Figure 6: SQuAD performance vs input sequence length. The compression ratio is fixed at $1/4$. Uses Vicuna 1.5 7B with 16k maximum sequence length against our SQuAD (train) task with 7 (default) to 63 confusion contexts to increase the sequence length. We believe the drop in performance at 3k tokens is an artefact of the RoPE scaling and fine-tuning procedure used to extend the context window of the Vicuna model.

Table 3: Needle in a haystack results, averaged over all depths and sequence lengths S within the specified range. See Figure A4 for full heatmap results and more details.

S Range	Compression	LM-∞	H$_2$O	SparQ
	1	**100%**	**100%**	**100%**
8k - 16k	1/4	23.5%	5.9%	**100%**
	1/8	11.8%	2.9%	**79.4%**
	1	**100%**	**100%**	**100%**
16k - 24k	1/4	23.5%	8.8%	**100%**
	1/8	11.8%	5.9%	**100%**
	1	**90.4%**	**90.4%**	**90.4%**
24k - 32k	1/4	23.5%	10.3%	**90.4%**
	1/8	11.8%	5.9%	**87.5%**

billion parameters. Results from the largest models are presented in Table 2, with further results in Figures A1 to A3.

We observe that SparQ Attention performance is robust across all tasks and models tested, as compression ratios of $1/2$ to $1/8$ are readily achievable with little to no loss in task performance. H$_2$O can attain good performance on some tasks such as TriviaQA and WikiTest-103, although other tasks, including SQuAD and Text Repetition, are more challenging and notable degradation occurs. LM-Infinite performance degrades across all tasks, demonstrating that the tasks do not permit the trivial solution of discarding the long input sequence.

5.3. Sequence Length Scaling

The sequence lengths of different examples in our main tasks vary between 1k and 2k tokens, whereas many LLMs support sequence lengths far greater than this. We considered two tasks to evaluate how SparQ Attention performs as sequence length scales.

The first task is a variation of SQuAD, which increases both sequence length and task difficulty by increasing the number of confusion contexts present in the prompt, which is akin to increasing the number of retrieved documents with a retrieval augmented generation system (Borgeaud et al., 2022). We test SparQ Attention and H$_2$O in this setting using Vicuna (Chiang et al., 2023), a descendent of Llama 2 that has been adapted for longer sequences. Both SparQ Attention and H$_2$O are configured to maintain a fixed compression ratio versus the dense baseline (keeping $r = 32$ and modifying k to maintain $1/4$ compression). The results in Figure 6 show that SparQ Attention is scalable to large sequences, as it can maintain performance up to 128k sequence length.

The second task evaluated is *needle in a haystack*, in which a "text needle" is inserted into the context at a certain depth, and the model is tasked with retrieving information from the needle. The exact implementation of this task we used is outlined in Dhinakaran (2024). We compare SparQ Attention with H$_2$O and LM-Infinite over a range of different compression ratios. The results, as seen in Table 3 and Figure A4, show that SparQ Attention achieves performance very close to the dense attention baseline, even in high-sparsity settings.
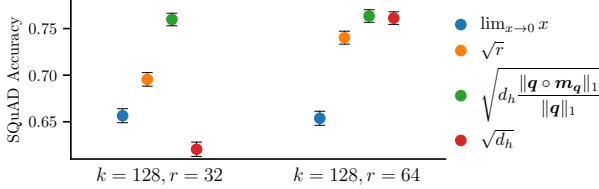
5.4. Ablations

Key cache compression The first step in SparQ Attention involves reading r components of the key cache to approximately determine which keys yield the highest attention scores. To examine the practical trade-off of the approximation, we look at how SparQ Attention performs when compared to a theoretical upper-bounding "*oracle*" which provides the exact top-k keys without requiring any data transfer to calculate the top-k. The results in Figure 7a show that SparQ Attention retains comparable performance to the oracle for a wide range of compression ratios, and attains considerably higher performance than a baseline compression scheme, in which a random low rank projection of $\boldsymbol{K}$ is transferred from memory.

Approximate softmax temperature To empirically support our statistical analysis of α agreement shown in Figure 4f, we evaluate a number of different viable temperature settings, including the square root of the head dimension ($\tau = \sqrt{d_h}$), the square root of the rank ($\tau = \sqrt{r}$), and the temperature proposed in Equation (9). We also consider the scenario where we do not reallocate mass to mean value ($\alpha = 0$), which corresponds to the limit of the temperature tending towards 0. We find that our proposed temperature performs best, as shown in Figure 7b.
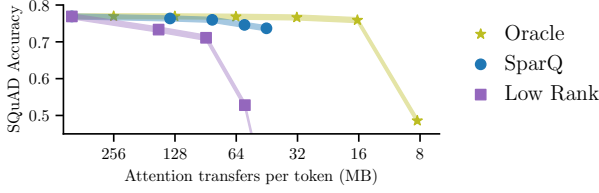
(a) Accuracy results of SparQ Attention and a random low rank compression scheme against an oracle top-k selector.

(b) Comparison of different softmax temperatures for approximate attention scores for two different hyperparameter configurations.
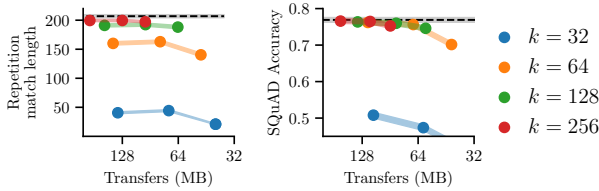
(c) Results for Repetition and SQuAD tasks with $r \in \{16, 32, 64\}$.

Figure 7: Ablations results, investigated on Llama 2 7B.

Hyperparameter selection The reduction of data transfer attained by SparQ Attention is controlled by its two hyper-parameters, k and r. Reducing either of these variables will improve the bandwidth efficiency, but can negatively impact task performance. Figure 7c shows the relationship between k and r on both of these factors. Based on these results, we propose a simple recipe of setting $k = 128$ and tuning r to maintain a good trade-off between data transfer and task performance for a range of models and tasks.

6. Benchmarking

The results above use a theoretical cost model of total memory transfers (the number of scalar elements transferred to and from memory per token), allowing us to evaluate SparQ Attention independently of the specific hardware setup and number formats used. To validate our findings, we performed a set of microbenchmarks of an attention operation in isolation, in addition to end-to-end performance benchmarks.

SparQ Attention benefits from two optimisations. The first is to store K twice, in both d_h-contiguous and S-contiguous layouts, since this allows for an efficient gather (indexing)
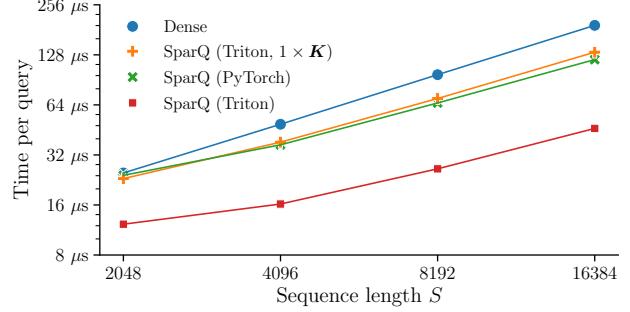

Figure 8: Microbenchmark results for batch size 64, 32 heads, $d_h = 128$, $r = 32$, $k = 128$, A100 (40GB).

Table 4: GPU microbenchmark performance with batch size 64, sequence length $S = 4096$, $r = 32$, $k = 128$. The theoretical speed-up is $6.4\times$.

Kernel	A100 (40GB)	A10G
Dense	49 μs ($1\times$)	128 μs ($1\times$)
SparQ (Triton, $1 \times K$)	38 μs ($1.28\times$)	79 μs ($1.63\times$)
SparQ (PyTorch)	37 μs ($1.33\times$)	78 μs ($1.63\times$)
SparQ (Triton)	**16 μs ($3.02\times$)**	**31 μs ($4.17\times$)**

on either axis, at the cost of 50% extra memory usage. The second optimisation is to use a fused gather-then-matmul operation to avoid writing the result of the gather to memory.

6.1. Microbenchmarks

We tested multiple implementations of baseline and SparQ Attention on IPU using the Poplar C++ interface and GPU using PyTorch (Paszke et al., 2019). In all cases, we used the Llama 2 7B shape parameters: 32 heads, $d_h = 128$. The implementations tested were: **Dense** baseline, choosing the faster of a plain PyTorch implementation and the builtin `scaled_dot_product_attention`, **SparQ (Triton)**, storing K twice and using fused gather-then-matmul kernels written using Triton (Tillet et al., 2019), **SparQ (PyTorch)**, with no Triton and **SparQ (Triton, $1 \times K$)**, storing K in d_h-contiguous layout only, for no additional memory cost. In an example configuration running on a single IPU from a Bow Pod$_{16}$, batch size 1, sequence length $S = 16384$, the dense baseline achieves 40.4 ms/query, while SparQ ($r = 32$, $k = 128$) achieves 5.28 ms/query for a speedup of $7.41\times$ (the theoretical speedup of SparQ is $7.53\times$). This near-perfect speedup is achieved because attention is strongly memory bound when using remote memory. In contrast, the baseline running in local SRAM takes 134 μs for a $345\times$ speedup, but this is only practically achievable when the whole model fits in SRAM.
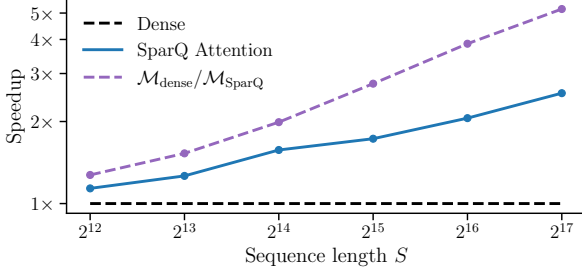
Figure 9: End-to-end CPU speedup results for SparQ Attention compared to the dense baseline with Llama 2 7B. This was achieved with batch size 1 and a compression ratio of $1/8$, with model weights represented in 8-bits and the KV cache represented in 16-bits. The $\mathcal{M}_{\text{dense}}/\mathcal{M}_{\text{SparQ}}$ line is a theoretical result, illustrating the upper bound of speedups attained with SparQ Attention.

Our achieved GPU speed-ups are presented in Table 4, and the performance trend with sequence length is shown in Figure 8. Standard error for all results given is $< 1\%$ of the mean. See Appendix F for further details.

These microbenchmark results show that the theoretical benefits of SparQ Attention can yield substantial wall-clock time speedups on current hardware. Further work is needed to show improvements for small batch sizes, and to investigate alternatives to storing K twice.

6.2. End-to-End Performance

In addition to the positive microbenchmark results, we further highlight the practical improvements SparQ Attention offers by benchmarking performance of the entire Transformer model on both CPU and GPU, implemented in `llama.cpp` and `gpt-fast` (Meta, 2023) respectively. In both cases, we measured the time it took to generate a single token, given an existing sequence length S.

CPU benchmarking We evaluated CPU benchmarking performance on AMD EPYC systems with up to 256GB memory. The results, as seen in Figure 9, show that SparQ Attention attains speedups at all sequence lengths evaluated, compared to the dense baseline. At the longest sequence lengths considered, SparQ Attention achieves a $2.5\times$ speedup, showcasing the benefits of reducing the data transfer associated with attention.

GPU benchmarking Our end-to-end GPU implementation is evaluated on a single H100 PCIe with 80GB memory. Despite utilising high-bandwidth memory, GPU inference still achieves end-to-end speedups when using SparQ Attention on modest sequence lengths, as seen in Figure 10.
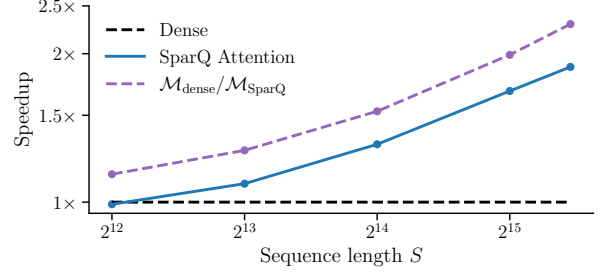

Figure 10: End-to-end H100 speedup results for SparQ Attention compared to the dense baseline with Llama 2 7B. These speedups were achieved with batch size 1 and a compression ratio of $1/8$, with both model weights and KV cache represented in 16 bits. The $\mathcal{M}_{\text{dense}}/\mathcal{M}_{\text{SparQ}}$ illustrates the theoretical maximum performance attainable under this setting.

7. Related Work

Efficient attention methods have been a very active area of research (Tay et al., 2020b). Schemes such as *Sparse Transformers* (Child et al., 2019), *Combiner* (Ren et al., 2021), *Longformer* (Beltagy et al., 2020), *BigBird* (Zaheer et al., 2020), *Reformer* (Kitaev et al., 2020) and *Sparse Sinkhorn Attention* (Tay et al., 2020a) have been developed to increase efficiency of the attention mechanism by extracting information from the most salient tokens in the sequence or approximating dense attention maps. Two schemes that reduce memory footprint and data transfer of the attention operation, while maintaining quadratic complexity are *Multi-Query Attention* (MQA) (Shazeer, 2019) and *Grouped-Query Attention* (GQA) (Ainslie et al., 2023) that share each KV head across multiple query heads. These methods form part of the architecture: they must be implemented during pre-training, carry varying task performance trade-offs, and may affect model quality and stability.

An emerging area of research similar to SparQ Attention aims to only adapt the inference procedure of a pre-trained model. The simplest method of this category is part of *FlexGen* (Sheng et al., 2023), and calculates exact attention scores, retrieving only the values associated with the top-k scores. This process uses the full key cache to produce attention scores, limiting the asymptotic reduction of the memory transfers to only 50%. LM-Infinite (Han et al., 2023) and StreamingLLM (Xiao et al., 2023) employ a fixed sparsity pattern preserving the most recent tokens and a few initial tokens for better attention efficiency, but are not selective in their cache lookup.

Eviction schemes cache only a subset of keys and values, by continually deleting tokens that are uninformative for future outputs. By reducing the cache size itself, both the

amount of memory used and data transferred are reduced. H_2O (Zhang et al., 2023), *Scissorhands* (Liu et al., 2023a) and *FastGen* (Ge et al., 2024) are examples of such eviction methods. H_2O uses a greedy eviction policy that maintains in memory the most salient "Heavy Hitter" tokens that contribute most to the attention scores. Scissorhands identifies and maintains "pivotal tokens" by counting when a token's attention score exceeds an importance threshold. FastGen adopts heuristics such as preventing the eviction of special tokens and punctuation, and tailors the compression strategy to each individual attention head. While these methods reduce the memory footprint of the KV cache as well as data transfer, they also lead to permanent loss of information from the context window, which can lead to mistakes for queries seeking less-attended parts of the sequence.

IceFormer (Mao et al., 2023) uses multiple existing approximate nearest neighbour algorithms for approximating attention scores of pre-trained models, focusing on speeding-up the prefill stage, rather than generation. *Scatterbrain* (Chen et al., 2021) employs similar techniques, but for computer vision applications.

In addition to compressing the KV cache, a number of methods strive to speed up LLM inference by inducing sparsity in the weights of the model. Deja Vu (Liu et al., 2023c) is a contextually-sparse approach that aims to predict which model parameters are required such that the error between the full computation and sparse approximation is minimised. Similarly, activation sparsity methods, including Kurtz et al. (2020) and Mirzadeh et al. (2024), exploit zero-values found in activations, typically induced by ReLU activation functions. Kurtz et al. (2020) introduce an alternative *forced activation threshold ReLU* function which can induce sparsity at specified thresholds. Similarly, Mirzadeh et al. (2024) replace the activation functions in LLMs with ReLUs, followed by additional fine-tuning. These methods are most suitable for small batch size and short sequence length regimes, where inference is bottlenecked by parameter transfer, rather than the KV cache, but are compatible with sparse attention techniques such as SparQ Attention.

An orthogonal line of work increases bandwidth efficiency by compressing the KV cache with 4-bit number formats (Liu et al., 2023b; Sheng et al., 2023). Liu et al. (2023a) demonstrate that 4-bit compression is complementary to techniques that reduce the number of transferred elements.

8. Conclusion

In this work we have presented SparQ Attention, a novel technique for unlocking faster inference for pre-trained LLMs. Our proposed technique modifies the attention mechanism to access only the relevant tokens from the KV cache at every generation step, leading to considerable data trans-

fer savings. This is particularly beneficial in long sequence length regimes, where inference speed is often bottlenecked by memory transfers rather than computation.

We also highlight the advantages of maintaining the full KV cache in memory for task performance by comparing SparQ Attention to other popular strategies which discard information from the input sequence. These alternative approaches rely on heuristics or predefined policies to determine which items in the KV cache to remove, which may not generalise across the wide range of applications LLMs are used for. We show that SparQ Attention is robust across numerous tasks and models, making it a viable technique for reducing inference times in unseen settings.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. GQA: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

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

Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., Van Den Driessche, G. B., Lespiau, J.-B.,

Damoc, B., Clark, A., et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pp. 2206–2240. PMLR, 2022.

Chen, B., Dao, T., Winsor, E., Song, Z., Rudra, A., and Ré, C. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34: 17413–17426, 2021.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., Stoica, I., and Xing, E. P. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Dhinakaran, A. The needle in a haystack test. `https://towardsdatascience.com/the-needle-in-a-haystack-test-a94974c1ad38`, February 2024.

Ge, S., Zhang, Y., Liu, L., Zhang, M., Han, J., and Gao, J. Model tells you what to discard: Adaptive kv cache compression for llms. In *International Conference on Learning Representations*, 2024.

Gerganov, G. llama.cpp. `https://github.com/ggerganov/llama.cpp`, 2024.

Graphcore. Bow-2000 datasheet. (Online: accessed 25 January 2024), March 2023. URL `https://docs.graphcore.ai/projects/bow-2000-datasheet`.

Han, C., Wang, Q., Xiong, W., Chen, Y., Ji, H., and Wang, S. LM-infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137*, 2023.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Karpathy, A. The unreasonable effectiveness of recurrent neural networks. (Online: accessed 27 January 2024), 2015. URL `https://github.com/karpathy/char-rnn`.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Kurtz, M., Kopinsky, J., Gelashvili, R., Matveev, A., Carr, J., Goin, M., Leiserson, W., Moore, S., Nell, B., Shavit, N., and Alistarh, D. Inducing and exploiting activation sparsity for fast neural network inference. In *International Conference on Machine Learning*, 2020.

Lin, C.-Y. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *arXiv preprint arXiv:2305.17118*, 2023a.

Liu, Z., Oguz, B., Zhao, C., Chang, E., Stock, P., Mehdad, Y., Shi, Y., Krishnamoorthi, R., and Chandra, V. LLM-QAT: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023b.

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., and Chen, B. Deja Vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, 2023c.

Mao, Y., Ester, M., and Li, K. Iceformer: Accelerated inference with long-sequence transformers on CPUs. In *Third Workshop on Efficient Natural Language and Speech Processing (ENLSP-III): Towards the Future of Large Language Models and their Emerging Descendants*, 2023.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., Pathak, S., Sifre, L., Rivière, M., Kale, M. S., Love, J., et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Meta. gpt-fast. `https://github.com/pytorch-labs/gpt-fast`, 2023.

Meta AI. Introducing meta llama 3: The most capable openly available llm to date. `https://https://ai.meta.com/blog/meta-llama-3`, April 2024 2024.

Mirzadeh, S. I., Alizadeh-Vahid, K., Mehta, S., del Mundo, C., Tuzel, O., Samei, G., Rastegari, M., and Farajtabar, M. ReLU strikes back: Exploiting activation sparsity in large language models. In *International Conference on Learning Representations*, 2024.

NVIDIA. NVIDIA A10 datasheet. (Online: accessed 22 January 2024), March 2022. URL https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a10/pdf/datasheet-new/nvidia-a10-datasheet.pdf.

NVIDIA. NVIDIA H100 datasheet. (Online: accessed 22 January 2024), July 2023. URL https://www.nvidia.com/en-gb/data-center/h100/.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

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, 2023.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training. (Online: accessed 29 January 2024), 2018. URL https://openai.com/research/language-unsupervised.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Ren, H., Dai, H., Dai, Z., Yang, M., Leskovec, J., Schuurmans, D., and Dai, B. Combiner: Full attention transformer with sparse computation cost. *Advances in Neural Information Processing Systems*, 34:22470–22482, 2021.

Roche, A., Malandain, G., Pennec, X., and Ayache, N. The correlation ratio as a new similarity measure for multimodal image registration. In *Medical Image Computing and Computer-Assisted Intervention — MICCAI'98: First International Conference Cambridge, MA, USA, October 11–13, 1998 Proceedings 1*, pp. 1115–1124. Springer, 1998.

Rosenblatt, M. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3):832 – 837, 1956.

See, A., Liu, P. J., and Manning, C. D. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

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

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 *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Su, J., Lu, Y., Pan, S., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021. URL https://arxiv.org/abs/2104.09864.

Tay, Y., Bahri, D., Yang, L., Metzler, D., and Juan, D.-C. Sparse sinkhorn attention. In *International Conference on Machine Learning*, pp. 9438–9447. PMLR, 2020a.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient transformers: A survey. *CoRR*, abs/2009.06732, 2020b. URL https://arxiv.org/abs/2009.06732.

Tillet, P., Kung, H.-T., and Cox, D. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

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*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vig, J. A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 37–42, 01 2019.

Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Yun, C., Chang, Y.-W., Bhojanapalli, S., Rawat, A. S., Reddi, S., and Kumar, S. $O(n)$ connections are expressive enough: Universal approximability of sparse transformers. *Advances in Neural Information Processing Systems*, 33:13783–13794, 2020.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297, 2020.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H$_2$O: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.

A. Detailed Results

Figures A1 to A3 report the compression/performance trade-off curves for all models and tasks that were evaluated.
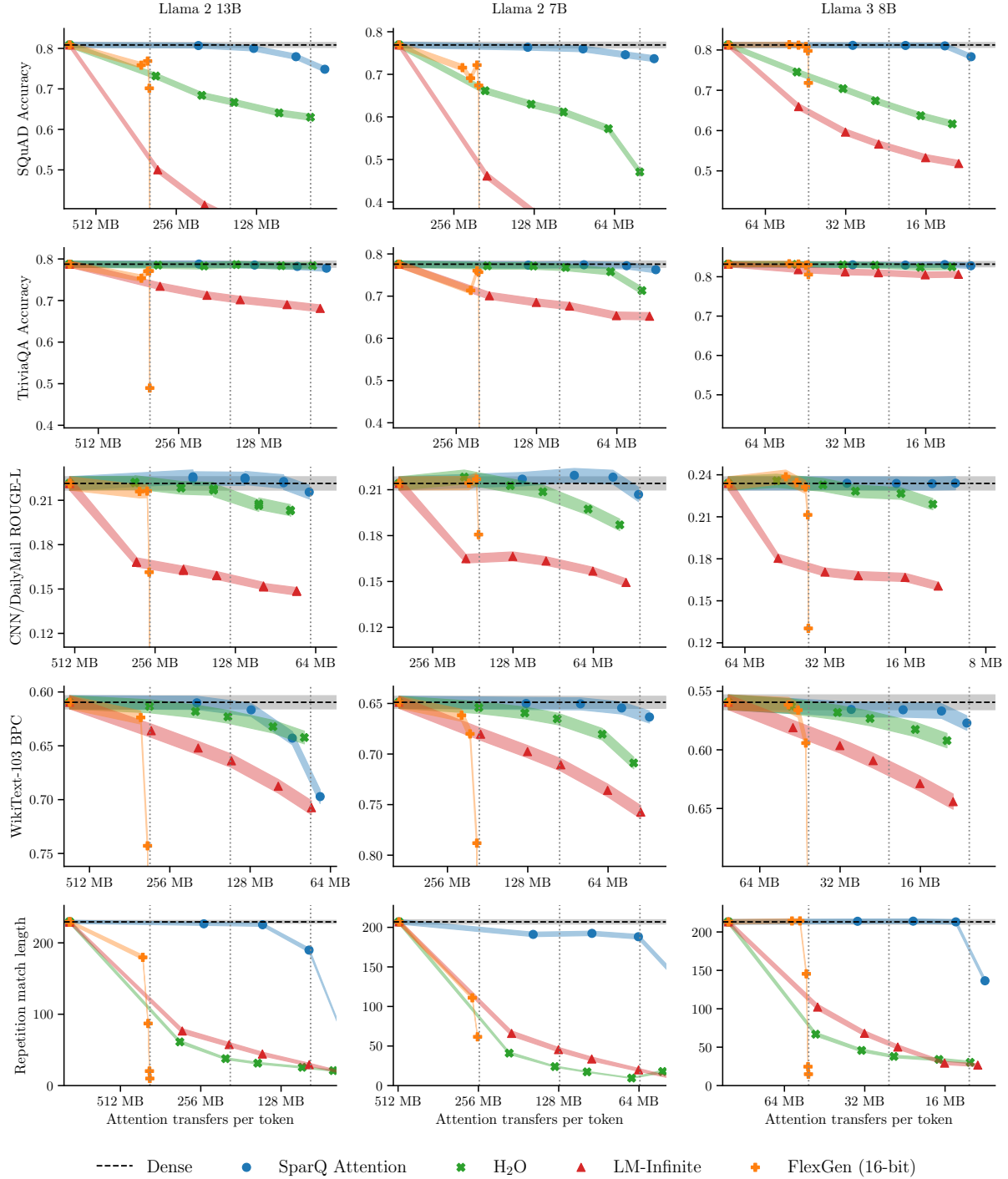

Figure A1: Compression versus performance trade-off curves over all tasks for the Llama 2 and Llama 3 model families. The y-axis minimum is set to (0.5, 0.5, 0.5, 1.25, 0.0)× the dense baseline for the tasks, reading top-to-bottom, in order to give a consistent view of the performance loss across models. Vertical dotted lines show $1/2\times$, $1/4\times$ and $1/8\times$ compression versus dense. Shaded lines show ±1 standard error of the mean (uncertainty due to a finite test set).
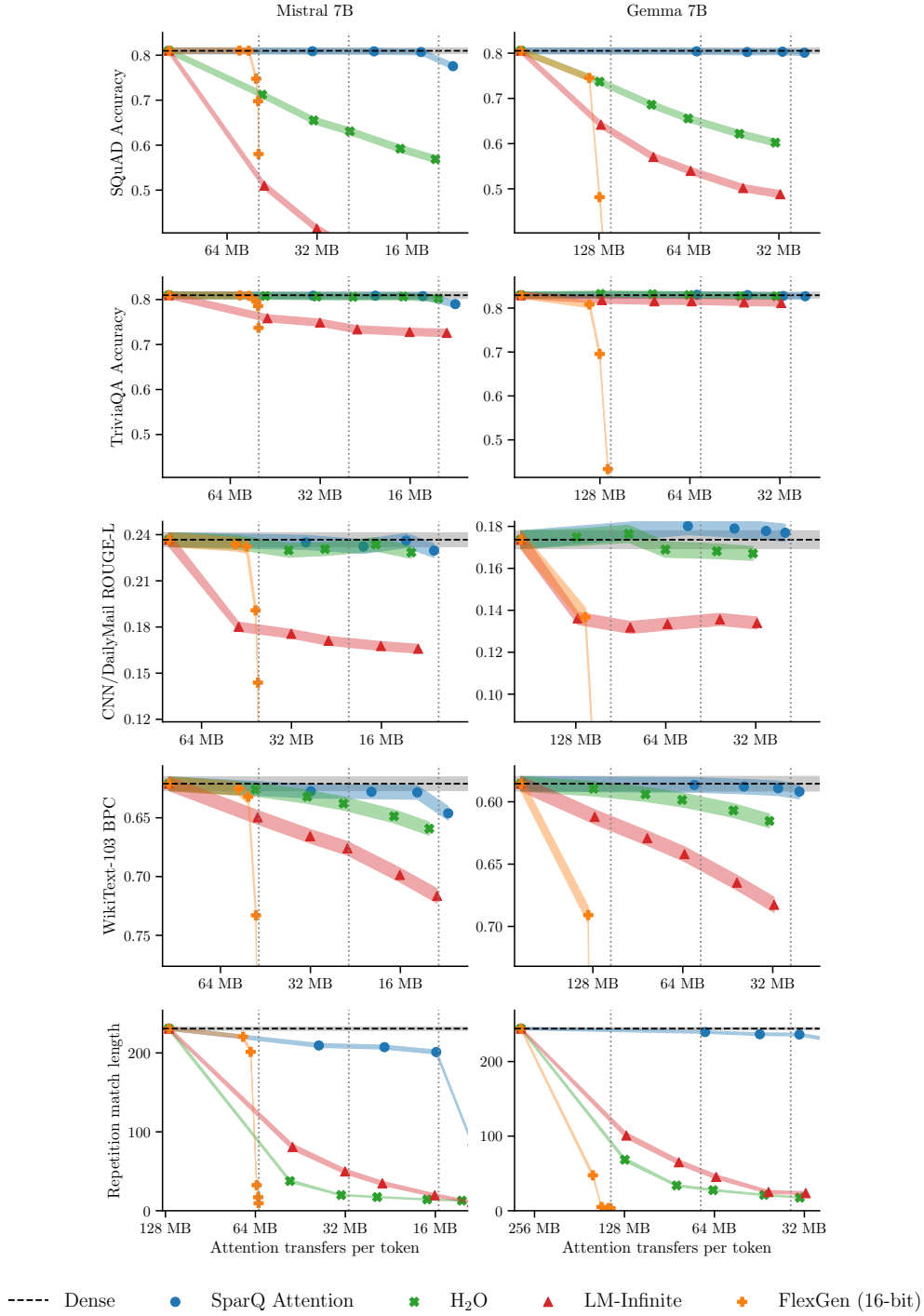
Figure A2: Compression versus performance trade-off curves over all tasks for Mistral 7B and Gemma 7B models. The y-axis minimum is set to $(0.5, 0.5, 0.5, 1.25, 0.0)\times$ the dense baseline for the tasks, reading top-to-bottom, in order to give a consistent view of the performance loss across models. Vertical dotted lines show $1/2\times$, $1/4\times$ and $1/8\times$ compression versus dense. Shaded lines show ± 1 standard error of the mean (uncertainty due to a finite test set).
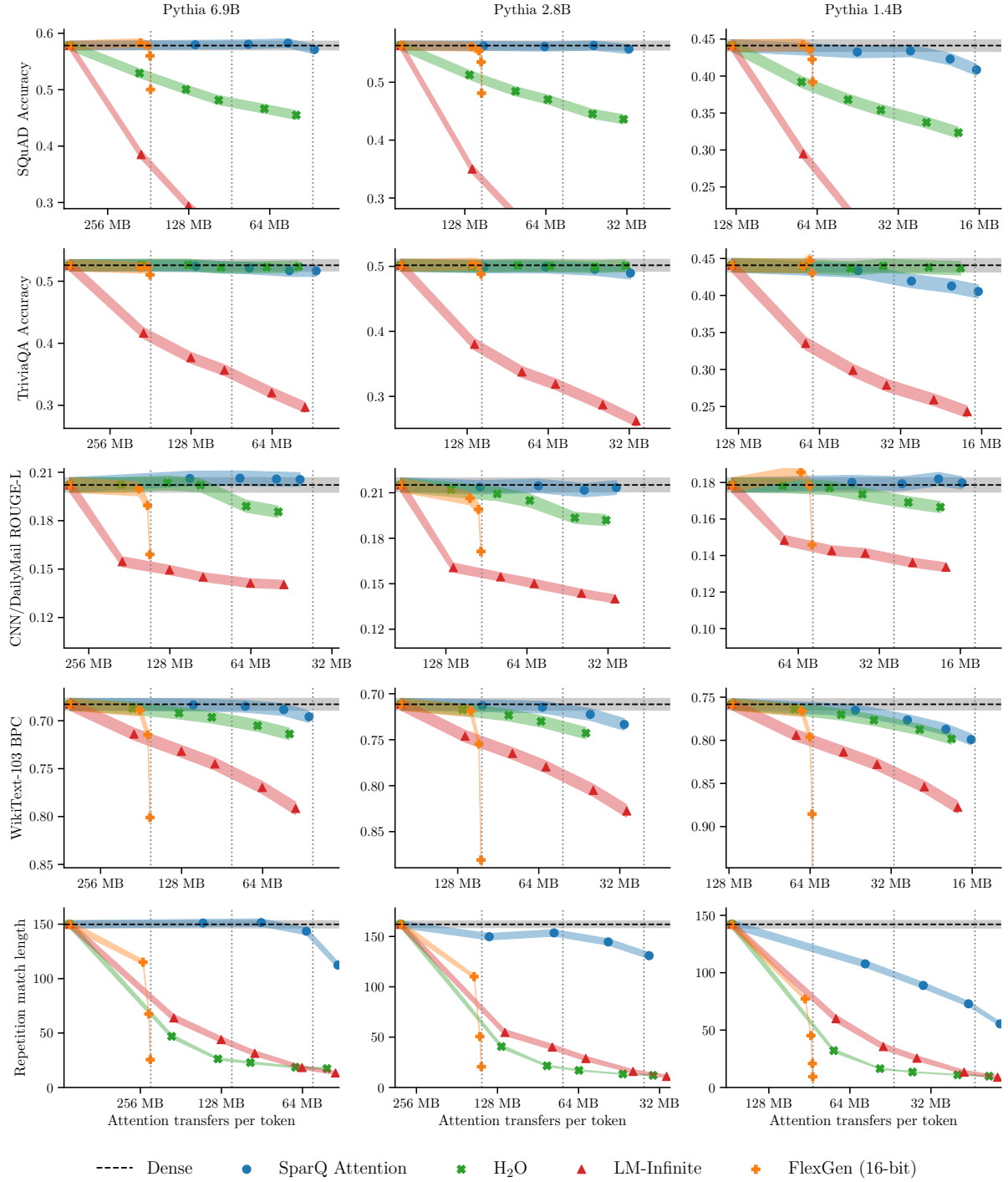
Figure A3: Compression versus performance trade-off curves over all tasks for the Pythia family of models. The y-axis minimum is set to $(0.5, 0.5, 0.5, 1.25, 0.0)\times$ the dense baseline for the tasks, reading top-to-bottom, in order to give a consistent view of the performance loss across models. Vertical dotted lines show $1/2\times$, $1/4\times$ and $1/8\times$ compression versus dense. Shaded lines show ± 1 standard error of the mean (uncertainty due to a finite test set).
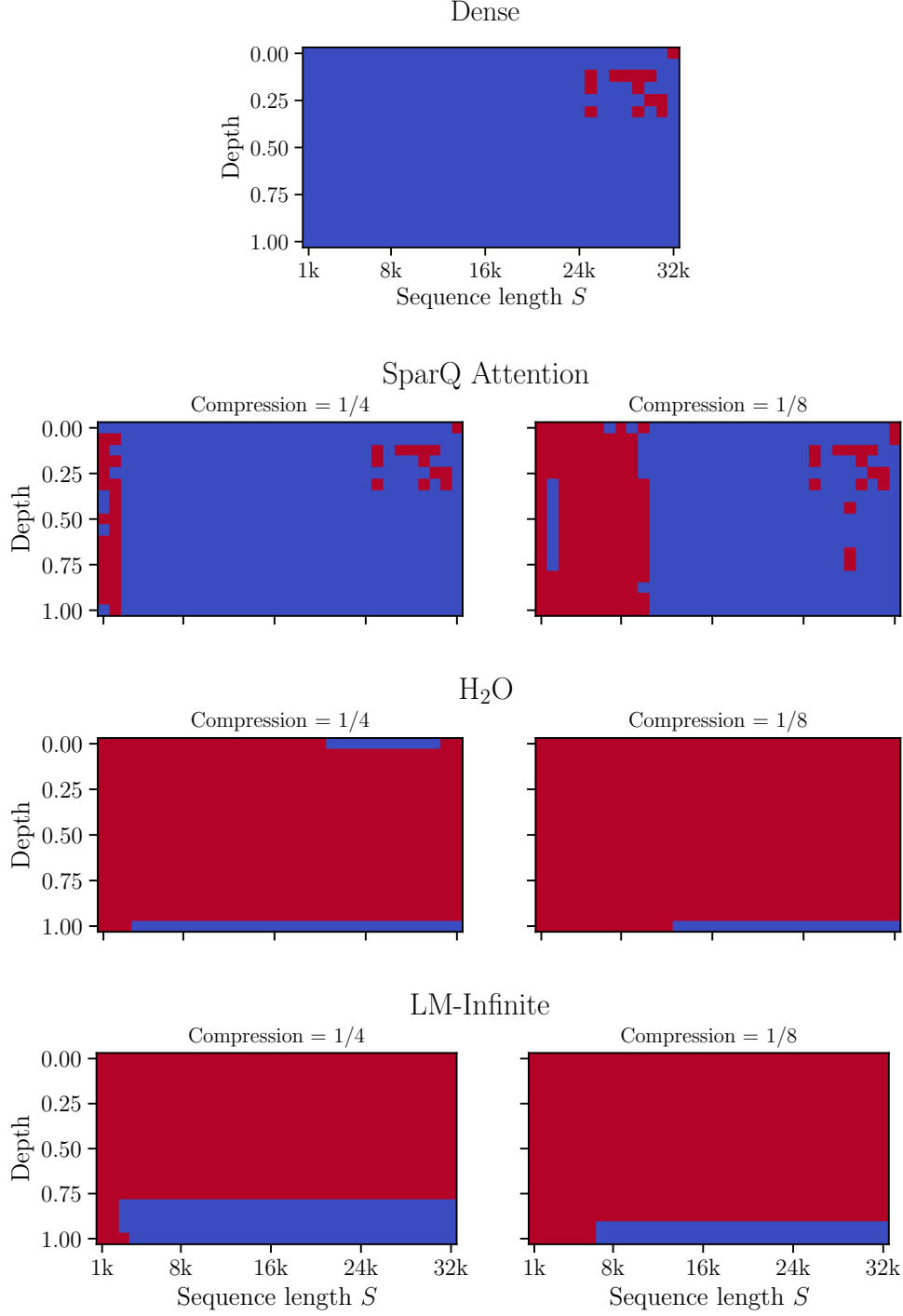
Dense

SparQ Attention

Compression = 1/4   Compression = 1/8

H₂O

Compression = 1/4   Compression = 1/8

LM-Infinite

Compression = 1/4   Compression = 1/8

Figure A4: Heatmaps of the results for the needle in a haystack task, as outlined in Dhinakaran (2024). For various sequence lengths (comprising of essays by Paul Graham), the needle (which is the sequence "*The best thing to do in San Francisco is eat a sandwich and sit in Dolores Park on a sunny day*") is inserted into the sequence at various depths, and the model is prompted to answer what the best thing to do in San Francisco is. The heatmaps show whether the model returns the correct (blue) or incorrect (red) answer. We evaluated this task over SparQ Attention, H₂O and LM-Infinite (in addition to the dense baseline) on the `togethercomputer/LLaMA-2-7B-32K` model, for two compression ratios.

B. Code

```python
from torch import softmax, sqrt, tensor, topk


def gather(t, dim, i):
    dim += (dim < 0) * t.ndim
    return t.gather(dim, i.expand(*t.shape[:dim], i.shape[dim], *t.shape[dim + 1 :]))


def attn(Q, K, V, M):
    s = (Q @ K.transpose(-1, -2)) / sqrt(tensor(Q.shape[-1])) + M
    y = softmax(s, dim=-1) @ V
    return y


def sparq_attn(Q, K, V, V_mean, M, r, k):
    # Q -- (batch_size, n_kv_heads, n_heads // n_kv_heads, 1, head_size)
    # K, V -- (batch_size, n_kv_heads, 1, seq_len, head_size)

    # 1. Approximate attention scores using r largest components of Q
    i1 = topk(abs(Q).sum(dim=2, keepdim=True), r, -1).indices
    Q_hat, K_hat = gather(Q, -1, i1), gather(K, -1, i1)
    scale = sqrt(
        Q.shape[-1]
        * abs(Q_hat).sum(dim=-1, keepdim=True)
        / abs(Q).sum(dim=-1, keepdim=True)
    )
    s_hat = softmax(Q_hat @ K_hat.transpose(-1, -2) / scale + M, dim=-1)

    # 2. Gather top k positions based on approximate attention scores & run attention
    i2 = topk(s_hat.sum(dim=2, keepdim=True), k, -1).indices
    iKV = i2[..., 0, :, None]
    K, V, M = gather(K, -2, iKV), gather(V, -2, iKV), gather(M, -1, i2)
    y_ = attn(Q, K, V, M)

    # 3. Estimate the total score of the top k, and interpolate with V_mean
    alpha = gather(s_hat, -1, i2).sum(-1, keepdim=True)
    y = alpha * y_ + (1 - alpha) * V_mean
    return y
```

C. Arithmetic Intensity

Consider a full transformer layer, with N parameters, batch size B, C elements in the attention KV cache per batch element and g grouped-query heads per key-value head. This implies the arithmetic intensity:

$$\frac{\mathcal{A}}{\mathcal{M}} = \frac{BN + BCg}{N + BC} = \frac{N + Cg}{N/B + C} \tag{C1}$$

We can increase arithmetic intensity by making B large, causing $\mathcal{A}/\mathcal{M}$ to approach $N/C + g$. Hence the limiting factor for large-batch transformer inference is the ratio of the KV cache size per-item to the size of the model.

We can alternatively express this in terms of the model's basic hyperparameters. A standard transformer with model dimension d_m and sequence-length S has $N = 12(d_m)^2$ and $C = 2\,S\,d_m/g$ (Kaplan et al., 2020). Substituting these values into Equation (C1), we get

$$\frac{\mathcal{A}}{\mathcal{M}} = \frac{6 + \rho\,g}{6/B + \rho} \tag{C2}$$

where $\rho = S/(gd_m)$ is a variable we have introduced, and underlies the KV cache-model size relationship outlined above, determining the point at which the model becomes memory bandwidth bound. We observe that the arithmetic intensity as batch size increases approaches $g + 6/\rho$.
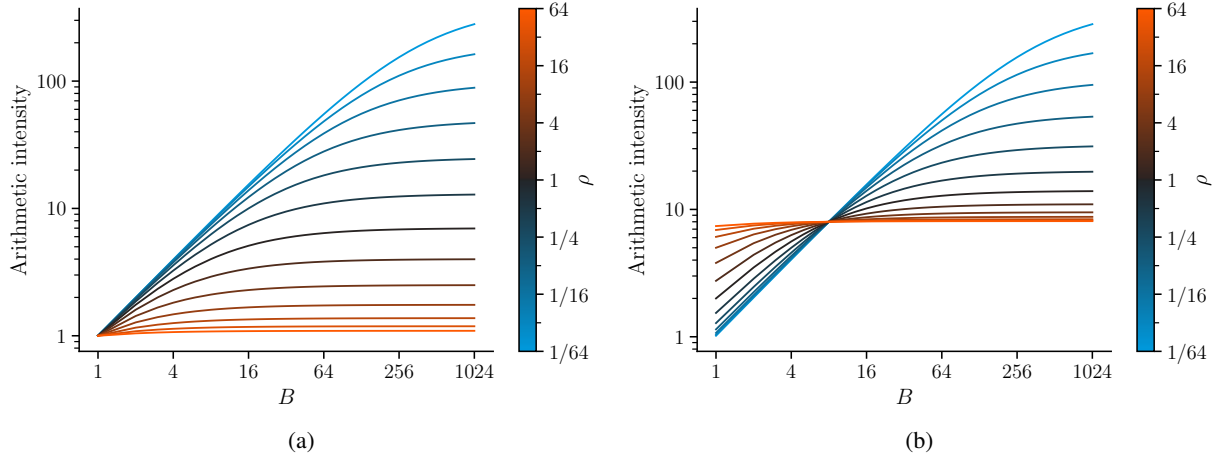
(a)

(b)

Figure C1: Relationship between $\rho = S/(gd_m)$, B and arithmetic intensity. (a) Multi-head attention, $g = 1$. (b) Grouped-query attention, $g = 8$. This highlights the importance of ρ, even with large batch size and GQA.

Model	g	d_m	S	$\rho = S/(gd_m)$	Max $\mathcal{A}/\mathcal{M}$
Llama 2 7B	1	4096	4096	1	7
Llama 2 70B	8	8192	4096	1/16	104
Llama 2 70B	8	8192	16384	1/4	32

The general relationship between ρ and arithmetic intensity is shown in Figure C1.

Hardware   Properties of selected machine learning hardware.[4] Note that $r_{\mathcal{A}}$ is the number of multiply-adds per second and $r_{\mathcal{M}}$ the number of data elements transferred per second.

Name	Memory technology	$r_{\mathcal{A}}/10^{12}$	$r_{\mathcal{M}}/10^{12}$	$r_{\mathcal{A}}/r_{\mathcal{M}}$
Bow IPU (FP16)	SRAM	175	5.5	32
A10 GPU (INT8)	GDDR	125	0.6	210
H100 SXM GPU (FP8)	HBM	990	3.35	295

Comparing $r_{\mathcal{A}}/r_{\mathcal{M}}$ for this hardware to the arithmetic intensity achievable for standard transformer models, it's clear that sequence generation will hit a data transfer bottleneck.

In summary, we have seen that sequence generation exhibits a large-batch arithmetic intensity of just 7 for multi-head attention with $S = d_m$, up to 100 for grouped-query attention with $S \ll d_m$, while ML hardware can provide $r_{\mathcal{A}}/r_{\mathcal{M}} > 200$.

D. Measuring Time Spent in Attention

Deriving exact measurements of how long is spent in attention layers in optimised inference libraries such as `llama.cpp` and `vLLM` can be non-trivial, due to limited existing tooling in their implementations and (to a lesser extent) probing models during inference may impact performance. `llama.cpp` features a benchmarking tool called `llama-bench`, which measures the time it takes to either prefill a model with a prompt of certain length, or autoregressively generate a sequence.

We employ `llama-bench`'s existing functionality to calculate the approximate time that is spent in attention, by observing that when generating a single token, the compute and data transfer associated with the attention layers scale linearly with the

---

[4]For IPU (Graphcore, 2023), we use the exchange memory bandwidth of 11 TB/s. A10 (NVIDIA, 2022). H100 (NVIDIA, 2023).
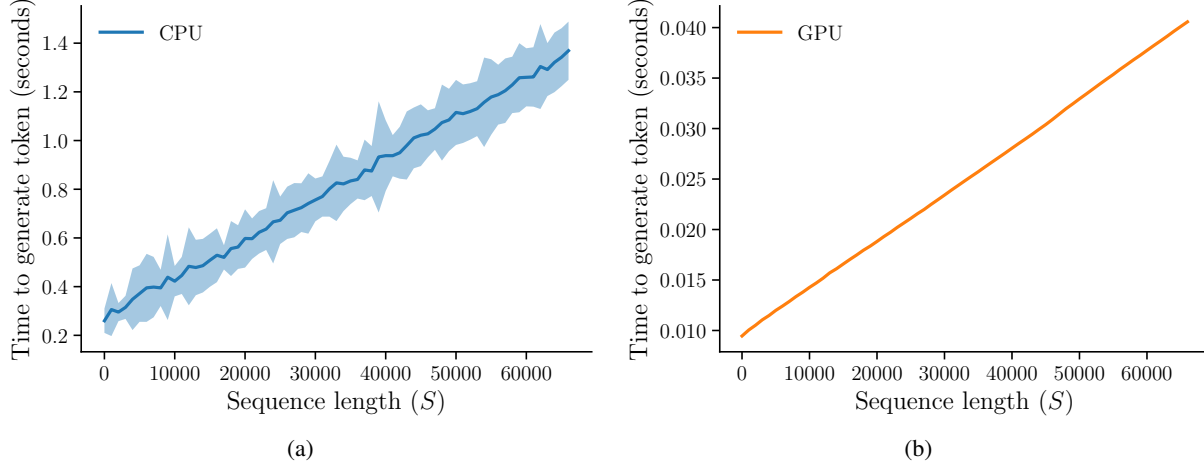
(a) (b)

Figure D1: For each sequence length S, we prefill the KV cache with a prompt of length S before measuring the time it takes to generate a single additional token with Llama 2 7B (batch size 1). We report the mean and standard deviation over 200 runs per sequence length, for (a) CPU and (b) A100 GPU. Measurements were carried out via the `llama-bench` tooling from `llama.cpp`.

sequence length, with all other costs remaining constant. This can be seen in Figure D1, which shows the measured time it takes to generate a single token, given an existing sequence length S. As S increases, the time it takes to generate a single token scales linearly.

From the measured benchmarks, lines of best fit were computed over the interquartile range of each S (to reduce variance) for each hardware platform, which were found to be

$$y_{\text{CPU}}(S) = (1.62 \times 10^{-5})S + 0.2438 \tag{D1}$$

and

$$y_{\text{GPU}}(S) = (4.686e \times 10^{-7})S + 0.009481 \tag{D2}$$

for CPU and GPU respectively. The value of $y_{\text{XPU}}(0)$ corresponds to the time it takes for all non-attention data transfer and operations. Therefore, the proportion of time spent in attention, $z_{\text{XPU}}(S)$, can be approximated as

$$z_{\text{XPU}}(S) \approx \frac{y_{\text{XPU}}(S) - y_{\text{XPU}}(0)}{y_{\text{XPU}}(S)}, \tag{D3}$$

the results of which can be seen in Figure 3.

E. Attention Sparsity Analysis

In order to understand how to approximate attention in pretrained transformers, we analysed the queries, values and intermediate *scores* vector (softmax output). We took 40 examples from our SQuAD 1-shot task, and generated the first completion token using the dense Llama 2 7B model, capturing the q vector and K, V matrices from every layer and attention head, showing derived statistics in Figures 4, E1 and E2.

In Figures 4c and 4d we show that elements of the query vectors are not normally distributed, but have high sample kurtosis values. If compared to a normal distribution, the combined mass of the elements with absolute *z-score* exceeding 3.0 is up to $20\times$ higher. This leads us to theorise that query vectors in a pre-trained model inherently encode information **sparsely** using the tails. Therefore, the magnitude based sparsity we induce in the first stage of the algorithm does not significantly harm the approximation of the attention mappings.

We validate this claim by comparing the correspondence between the exact and approximated attention scores. SparQ Attention uses the approximate attention scores to only choose the tokens that are important for the next generation step. The
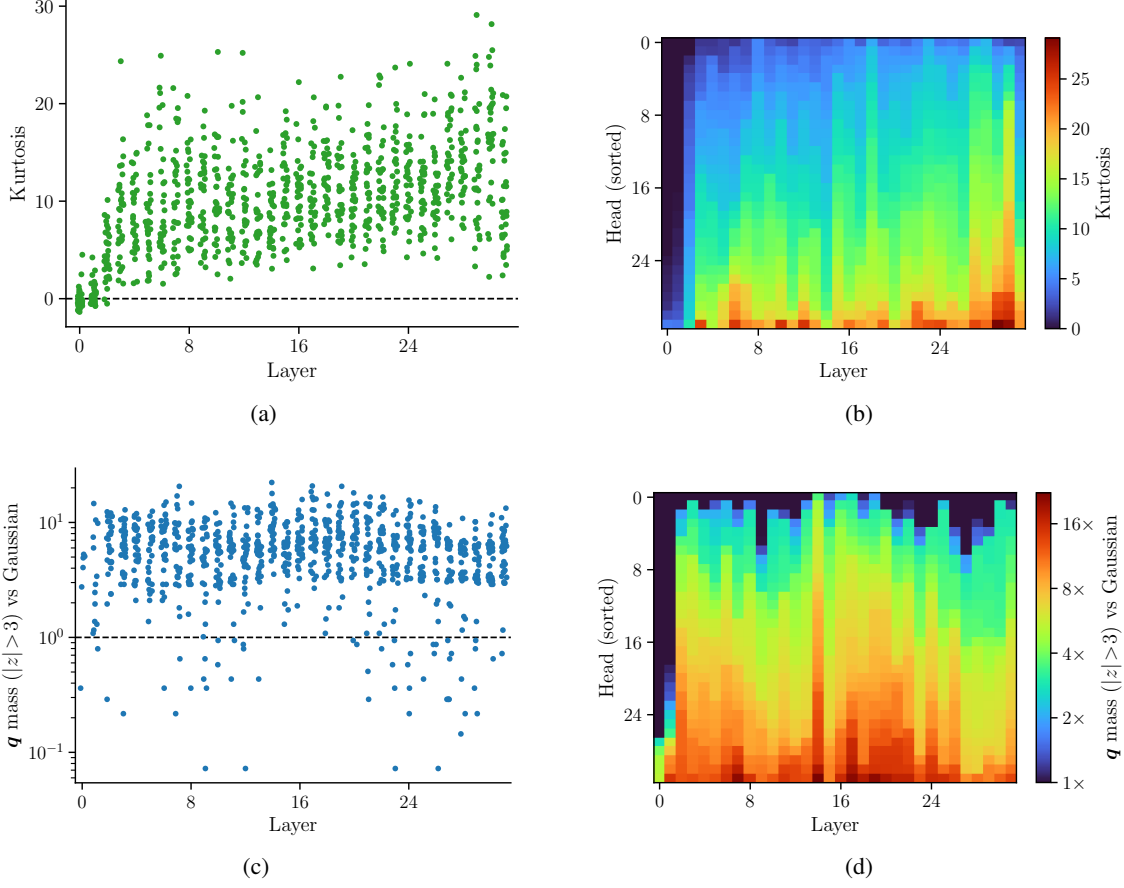
(a)

(b)

(c)

(d)

Figure E1: Statistics of components of q for each head, as a function of layer. (Top) Kurtosis (Fisher), indicating that most heads have heavy-tailed q. (Bottom) z-value mass, normalised by that of a Gaussian (0.3%), showing that most heads are outlier-heavy. All Llama 2 7B, measured over 40 SQuAD examples.

actual values of the approximate scores are not relevant, as these scores are not multiplied with value vectors and thus the property of interest to us is whether the top-k indices in the approximate scores match those of the exact counterpart. This can be measured on a scale from 0 to 1, where 1 means top-k indices are identical between the approximation and the exact scores and 0 means these sets do not overlap. We call this measure *top-k correspondence*. Figure E2 provides an overview how the choice of rank and k affects the top-k correspondence aggregated over all attention heads of the model. We see that the query vector sparsity of 50% and 75% maintain high top-k correspondence to the exact attention scores, which is consistently maintained over various values of k. Further analysis and a more detailed examination of top-k correspondence is presented in Appendix E.

It is useful to drop positions in V given attention scores, but this can save at most half of the data transfers, since the whole of K is needed to calculate these scores. We propose approximating these scores using a subset of the components of K. To test such an approximation, we measure the proportion of *overlap* between the top 32 positions in the approximated and true scores. If overlap is high, we can use the approximation to avoid transferring the whole K matrix, instead only transferring some components of K for all positions, then all components of K for some positions.

Our hypothesis is that the r largest-magnitude components of q are most useful to predicting the score, $qK^\top$. The coverage of this technique against an arbitrary-component baseline is shown in Figure E2. These results show that it is possible to achieve reasonably high overlap even using $r = d_h/8$, but that some later layers are harder to predict. Using the top-r components outperforms the first r baseline considerably.
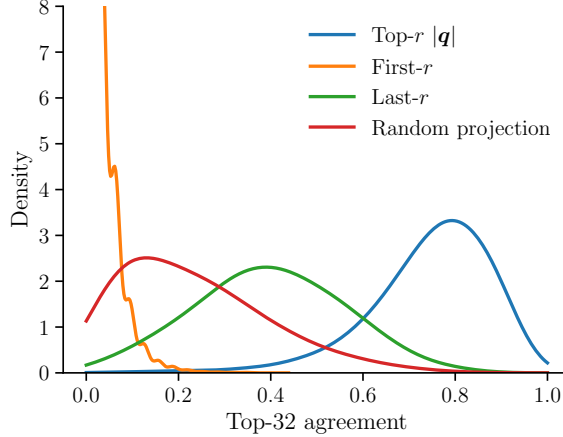
Figure E2: Top-k agreement between approximate and true scores (Llama 2 7B, measured over 40 SQuAD examples). Top-k agreement is the proportion of the top-k positions that are correctly predicted by an approximated softmax, using a projection of $\boldsymbol{q}$, either component-wise or a random low-rank projection.

F. Benchmarking Detail

Benchmarking code is made available from:
https://github.com/graphcore-research/llm-inference-research/tree/2024-05-sparq.

IPU measurements We tested custom fully-fused Poplar implementations of both dense attention and SparQ Attention, compiled using Poplar SDK 3.3.0+1403. On initialisation, we fill large $\boldsymbol{K}$ and $\boldsymbol{V}$ tensors with values $\sim N(0,1)$ in streaming memory. On each benchmarking (outer) iteration, we first randomise the contents of a $\boldsymbol{q}$ in local memory, then perform multiple inner repeats of the attention op being profiled. We use 4 inner repeats for dense attention, otherwise $1024/\text{batch\_size}$, chosen because dense attention is much slower, and we swept a wide range of settings. We ran an outer loop of 2 warm-up iterations followed by 10 timed iterations, reporting the mean and standard error. The sweep covered $S \in [1024, 2048, \ldots, 65536]$, $\text{batch\_size} \in [1, 4, 16, 64]$, SparQ Attention $r \in [16, 32, 64]$ and $k \in [64, 128, 256, 512]$.

GPU measurements All experiments use PyTorch 2.1.2+cu121 on Ubuntu AWS instances. To set up the experiment, we initialise the large $\boldsymbol{K}$ and $\boldsymbol{V}$ tensors with values $\sim N(0,1)$. On each step, we draw $\boldsymbol{q} \sim N(0,1)$, run `torch.cuda.synchronize` before starting a host-side wall-clock timer, run the op, and synchronize again before stopping the timer. We run 20 warm-up iterations followed by 200 timed iterations, reporting mean and standard error. For dense baseline implementations, we tested a vanilla PyTorch implementation, with/without `torch.compile` and `torch.nn.functional.scaled_dot_product_attention`, selecting each backend (math, flash, mem_efficient) manually. For SparQ Attention implementations, we tested vanilla PyTorch (lightly hand-optimised from Appendix B), with/without `torch.compile`. We also toggled fused gather-matmul kernels written in Triton, and whether $\boldsymbol{K}$ was stored twice in S-contiguous (for **Step 1**) and d_h-contiguous (for **Step 2**) layouts, or only once in d_h-contiguous layout. We tested $S \in [1024, 2048, 4096, 8192, 16384]$, $\text{batch\_size} \in [1, 4, 16, 64]$, SparQ Attention $r \in [16, 32, 64]$ and $k \in [64, 128, 256, 512]$.

Additional results In addition to the headline results shared in Section 6 and Figure 8, we give an aggregate picture of the trends in Figure F1. Since the number and dimension of heads is fixed, the x-axis is proportional to the size of the input tensors. On IPU (M2000), strong speedups are available across a range of input sizes, principally depending on r, but also on k (not shown). On GPU, sufficient input size is required to observe a speedup over the dense baseline, with the more bandwidth-limited A10G reaching speedups sooner. While part of this effect can be linked to the fundamental additional complexity of SparQ Attention, we anticipate that small input sizes could be accelerated considerably with additional kernel fusion. With an appropriate limit to sequence length, SparQ Attention could even be fused into a single CUDA kernel.
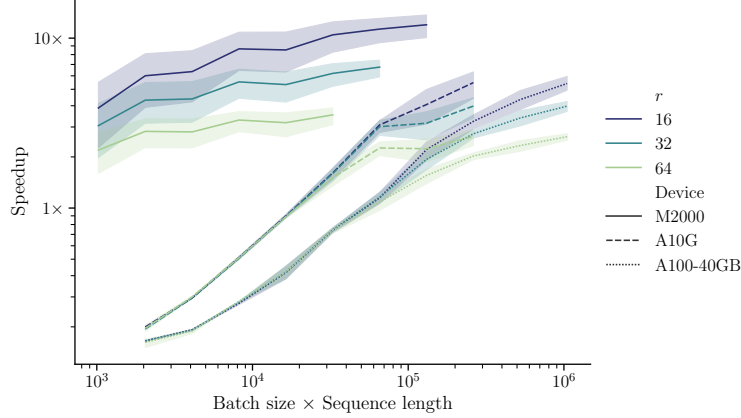
Figure F1: SparQ speedup over the dense baseline, across a range of batch size (1-64), sequence length (1024-65536) and k (64-512), for different devices. We note that for both GPUs, the number of KV elements is a limiting factor for the achieved speedup, and that this could be improved by writing a fully fused SparQ Attention kernel.

Storing K twice   One limitation of a theoretical model of data transfer is that it does not account for the granularity of memory access. Since the K matrix is indexed on different axes in **Step 1** and **Step 2** of SparQ Attention, a naive implementation would fetch non-contiguous elements in one of the two steps. To mitigate this, we propose storing K twice, once in S-major format and once in d_h-major format. This increases KV cache memory usage by 50%, but uses only a small amount of extra bandwidth to write k twice. This extra write is non-contiguous, but small, so should not form a bottleneck.

G. Methodology

We provide a comprehensive description of our experimental setup for reference in Table G1.

Baselines   We use our own implementation of H_2O (Zhang et al., 2023), which differs from the authors' implementation in that it uses a fixed cache size k, rather than a fixed proportion of the current sequence length. To validate that these implementations are sufficiently similar, we ran their implementation through our harness on a small model and sample size. On SQuAD 1-shot, with Pythia-1.4B, using $k = 256$, $l = 64$, our implementation was correct for 60 of 200 examples, theirs for 57 (the dense baseline achieved 74). Perhaps more importantly, we found that of the 79 times that either output differed from dense, 41 occurrences showed a 20-character prefix match between our implementation and theirs. The fact that the two implementations often generate the same errors (despite minor implementation differences) reassures us that our results should be a fair representation of H_2O.

Compression ratio   We define the compression ratio as the ratio of attention data transfers required for the sparse technique and the dense data transfers. Similarly as we have derived the transfers for SparQ Attention in Section 4, we can define the transfers required for each baseline technique:

$$\mathcal{M}_{\text{H}_2\text{O}} = 2\,k\,d_h + 2\,d_h + 2\,S$$
$$\mathcal{M}_{\text{LM}_\infty} = 2\,k\,d_h + 2\,d_h$$
$$\mathcal{M}_{\text{FlexGen}} = S\,d_h + k\,d_h + 2\,d_h$$

and $\mathcal{M}_{\text{dense}} = 2\,S\,d_h + 2\,d_h$. For each technique, the compression ratio is then $\mathcal{M}_{\text{technique}}/\mathcal{M}_{\text{dense}}$.

Models	Llama 2	7B ($d_h = 128$, Max $S = 4096$, $g = 1$)
		13B ($d_h = 128$, Max $S = 4096$, $g = 1$)
	Llama 3	8B ($d_h = 128$, Max $S = 8192$, $g = 4$)
	Mistral	7B ($d_h = 128$, Max $S = 8192$, $g = 4$)
	Gemma	7B ($d_h = 256$, Max $S = 8192$, $g = 1$)
	Pythia	1.4B ($d_h = 128$, Max $S = 2048$, $g = 1$)
		2.8B ($d_h = 80$, Max $S = 2048$, $g = 1$)
		6.9B ($d_h = 128$, Max $S = 2048$, $g = 1$)
Tasks	Question Answering	SQuAD 1-shot (4000 samples)
		TriviaQA 0-shot (2992 samples)
	Summarisation	CNN/DailyMail 0-shot (500 samples)
	Language Modelling	WikiText-103 LM (500 samples)
	Artificial	Repetition (1000 samples)
Baselines	H$_2$O	keep $(k - l)$ tokens with highest $\text{score}(n) = \sum_i s_{in}$ and the most recent $l = k/4$ $k \in \{192, 256, 384, 512, 768\}$
	LM-Infinite	take the first 16 tokens, and most recent $k - 16$ $k \in \{192, 256, 384, 512, 768\}$
	FlexGen	take top-k tokens using exact attention scores $k \in \{2, 8, 32, 128, 256\}$
SparQ Attention	Rank r	$\{8, 16, 32, 64\}$
	Number of values k	128
	Local window l	$k/4$

Table G1: Experimental setup.

G.1. Examples

We illustrate the task setup with a single example per task, showing the prompt formatting and a cherry-picked example. In each case, we show outputs from a dense Llama 2 13B model, SparQ Attention ($r = 8, k = 128$), H_2O and LM-Infinite ($k = 192$). Where "..." appears, we have truncated the line of text for brevity.

G.1.1. QUESTION ANSWERING (SQuAD 1-SHOT)

```
### PROMPT (5725e1c4271a42140099d2d9)
Title: University of Chicago. Background: Current ...
Title: Harvard University. Background: Harvard has...
Title: Oxygen. Background: In one experiment, Lavo...
Title: Oxygen. Background: Oxygen storage methods ...
Title: Fresno, California. Background: This vibran...
Title: Fresno, California. Background: Before Worl...
Title: Steam engine. Background: The working fluid...
Title: Sky (United Kingdom). Background: While BSk...
From what you've just read about Fresno, California, please answer the following questions.
Question: Where is Audra McDonald from?
Answer: Fresno
Question: In what year did Roger Rocka's Dinner Theater & Good Company Players open?
Answer:

### OUTPUT
      DENSE: 1978
      SPARQ: 1978
        H2O: 1979
LM-INFINITE: 1975 (Roger Rock
```

G.1.2. QUESTION ANSWERING (TriviaQA 0-SHOT)

```
### PROMPT (dpql_5685)
Apéritifs and digestifs ( and) are drinks, typical...
Apéritifs
An apéritif is an alcoholic beverage usually serve...
"Apéritif" may also refer to a snack that precedes...
"Apéritif" is a French word derived from the Latin...
...
...
* Distilled liquors (ouzo, tequila, whisky or akva...
* Liquor cocktails (Black Russian, Rusty Nail, etc...
In certain areas, it is not uncommon for a digesti...
Bitter digestifs typically contain carminative her...
In many countries, people drink alcoholic beverage...
Question: Which aperitif is named for the Paris chemist who created it in 1846?
Answer:

### OUTPUT
      DENSE: Dubonnet
      SPARQ: Dubonnet
        H2O: Dubonnet
LM-INFINITE: Byrrh
```

Note that for Pythia, the prompt "Single-word answer:" was used in place of "Answer:", as this helped prevent the model from restating the question in the answer (often qualitatively correct, but not a regex match).

G.1.3. SUMMARISATION (CNN/DAILYMAIL)

```
### PROMPT (a62bbf503be06e8b1f8baa4f3cd537310d5aa3bc)
Article: Prince William arrived in China tonight for one of the most high-profil...
Summary:

### OUTPUT
      DENSE: Prince William arrived in China tonight for one of the most high-profile { ...
      SPARQ: Prince William arrived in China tonight for one of the most high-profile { ...
        H2O: Prince William arrived in China tonight for a high-profile visit that will ...
LM-INFINITE: Prince William and Kate Middleton are in Japan for a three-day tour. The ro...
```

G.1.4. REPETITION (SHAKESPEARE)

```
### PROMPT (210496)
 you mistake me much;
I do lament the sickness of the king.
...
...
Peace, children, peace! the king doth love you well:
Incapable and shallow innocents,
You cannot guess who caused your father's death.

Boy:
Grandam, we can; for my good uncle Gloucester
Told me, the king, provoked by the queen,
Devised impeachments to imprison him :
And when my uncle told me so, he wept,
And hugg'd me in his arm, and kindly kiss'd my cheek;
...
...
 the king doth love you well:
Incapable and shallow innocents,
You cannot guess who caused your father's death.

Boy:
Grandam, we

### OUTPUT
      DENSE: can; for my good uncle Gloucester
      SPARQ: can; for my good uncle Gloucester
        H2O: can;
LM-INFINITE: 'll not stand to prate, but to the purpose.
```

G.1.5. LANGUAGE MODELLING (WIKITEXT-103)

```
### QUERY (2)

 = Mellor hill fort =

 Mellor hill fort is a prehistoric site in North West England , that dates from ...

 = = Location = =

 Mellor lies on the western edge of the Peak District in the Metropolitan Boroug...

 = = Background = =

 Until the 19th century little was known about hill forts ; none had been excava...
 The study of hill forts was popular in the 19th century , with a revival in the...

 = = History = =

 There is evidence of human activity on the site pre @-@ dating the Iron Age , a...
 A flint dagger was discovered on the site . This type of artefact is rare in Gr...
 The hill fort was built in and used throughout the Iron Age , as demonstrated b...

 Fragments of glass , possibly Roman in origin , and shards of pottery which date to the 1st and
2nd centuries AD , indicate the site was used in the Romano @-@ British period .  However no Roman
structures have been discovered , and the nature of Roman activity at the site is a source of
speculation .  The position of the hilltop indicate that it was easily defended ; however , local finds
indicate it was a high @-@ status settlement rather than a military outpost unless a similar feature
was located nearby .  One reason that Roman structures have not been identified is that the Romano

### BPC
      DENSE: 0.669
      SPARQ: 0.673
        H2O: 0.685
LM-INFINITE: 0.692
```