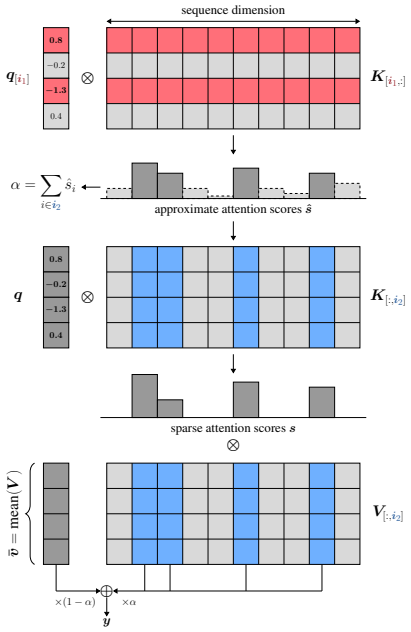
SPARQ ATTENTION: BANDWIDTH-EFFICIENT LLM INFERENCE

Luka Ribar*, **Ivan Chelombiev***, **Luke Hudlass-Galley***
Charlie Blake, **Carlo Luschi**, **Douglas Orr**
Graphcore Research
London, UK
{lukar,ivanc,lukehg,charlieb,carlo,douglaso}@graphcore.ai

ABSTRACT

The computational difficulties of large language model (LLM) inference remains a significant obstacle to their widespread deployment, with long input sequences and large batches causing token-generation to be bottlenecked by data-transfer. For this reason, we introduce **SparQ Attention**, a technique for increasing LLM inference throughput by utilising memory bandwidth more efficiently within 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. By evaluating Llama 2, Mistral and Pythia models on a wide range of downstream tasks, we show that SparQ Attention brings up to $8\times$ savings in attention data-transfer without substantial drops in accuracy.



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 $|\boldsymbol{q}|$
$\boldsymbol{i}_1 \leftarrow \text{argtopk}\left(|\boldsymbol{q}|, r\right)$

 # 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)
$\hat{\boldsymbol{s}} \leftarrow \text{softmax}\left(\boldsymbol{q}_{[\boldsymbol{i}_1]} \cdot \boldsymbol{K}^\top_{[\boldsymbol{i}_1,:]} / \tau\right)$

 # 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
$\boldsymbol{i}_2 \leftarrow \text{argtopk}\left(\hat{\boldsymbol{s}} + \boldsymbol{m}, k\right)$

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

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

 # 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 1: 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 $\boldsymbol{q}$ 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 approximate scores 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 $\overline{\boldsymbol{v}}$.

The full paper is available at https://arxiv.org/abs/2312.04985.

1 INTRODUCTION

Transformer models trained on large corpora of text have recently shown remarkable capabilities in solving complex natural language processing tasks (Kaplan et al., 2020; Brown et al., 2020). With the dramatic increase in model scale, LLMs became useful generalist tools that can be used for a multitude of text-based tasks due to *in-context learning* capabilities that emerge in LLMs at scale. These capabilities are unlocked by incorporating information and instructions through textual prompts (Wei et al., 2022), which are absorbed during generation as part of the attention cache without modifying the model weights. Therefore, one of the key obstacles to efficient inference deployment of LLMs remains their high memory bandwidth requirement when processing long sequences (Pope et al., 2023), i.e. long instruction prompts, lengthy chat histories or information retrieval from documents.

The main bottleneck appears due to the auto-regressive nature of transformer inference: the output is generated token by token, and for each model call, the full previous state (i.e. the *key-value cache*, or *KV cache*) needs to be fetched from memory. The size of the KV cache scales linearly with the sequence length as well as the batch size, thus rendering inference using long sequence lengths increasingly memory-bandwidth limited.

However, tokens generally only attend to a small part of the sequence at a time (Vig, 2019; Yun et al., 2020); thus, if it were possible to efficiently predict the tokens that will have high attention scores, memory traffic could be significantly reduced by only transferring the key and value pairs of high-scoring tokens. Building upon this idea, we present **SparQ (Sparse Query) Attention**, a technique for significantly reducing the memory bandwidth requirements of transformer inference. By approximating attention scores using a subset of query/key vector components, we fetch only the most relevant tokens for each inference step, greatly decreasing memory traffic without affecting task performance.

In order to evaluate the effectiveness of our technique, we curate a selection of downstream tasks that aim to test the model's abilities to effectively utilise the information within the provided textual context. This setup allows us to evaluate the trade-off between task performance and data transfer reduction, as well as compare different sparse attention techniques with respect to memory transfer efficiency. Thus, we clearly show how SparQ Attention performs favourably compared to state of the art and can lead up to $8\times$ compression with no loss in accuracy.

2 SPARQ ATTENTION

Consider a standard attention block and its associated memory requirements. A single head within an attention layer has a head dimension d_h, and processes an input token sequence of sequence length S. During token-by-token inference the output of an attention head is calculated as:

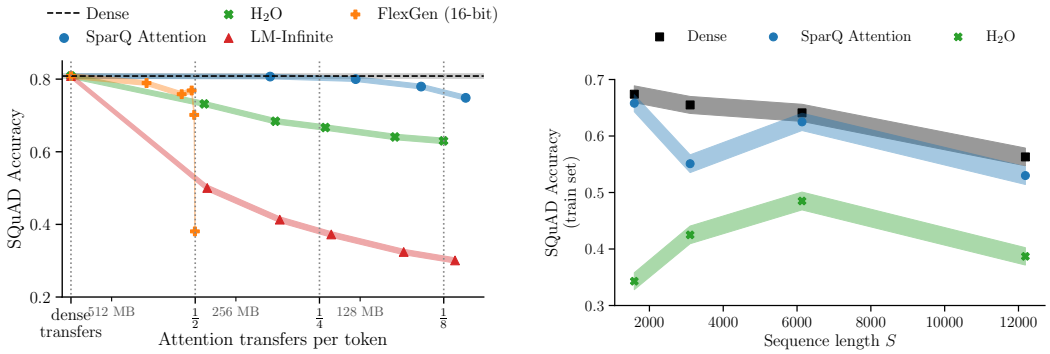
$$\boldsymbol{y} = \mathrm{softmax}\Big(\frac{\boldsymbol{q} \cdot \boldsymbol{K}^\top}{\sqrt{d_h}}\Big) \cdot \boldsymbol{V} \tag{1}$$

where $\boldsymbol{q}$ is the query, and $\boldsymbol{K} \in \mathbb{R}^{S \times d_h}$ and $\boldsymbol{V} \in \mathbb{R}^{S \times d_h}$ are the key and value caches respectively.

The total data transfer for transformer inference includes both the KV cache and model parameters. Whereas parameter transfers can be amortised over a large batch of sequences, the KV cache scales with sequence length and batch size, creating an efficiency bottleneck (see Appendix C).

To facilitate an accurate approximation of attention, without transferring the entire $\boldsymbol{K}$ and $\boldsymbol{V}$ matrices, we make the following observations (more details can be seen in Appendix G):

- The output of the $\mathrm{softmax}$ function in Equation (1) is dominated by a small number of components, with most of the components being close to 0.
- The indices of the largest attention scores can be efficiently predicted without fetching the full $\boldsymbol{K}$ matrix by *sparsifying* the query vector $\boldsymbol{q}$, keeping the r largest components).



(a) (b)

Figure 2: (a) 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 multiple models and tasks, shown in Figures A1 and A2. (b) 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.

Using these observations, we propose **SparQ Attention** (see Figure 1), comprising of three steps:

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

Step 2: Find the top-k positions in $\hat{\boldsymbol{s}}$ and fetch the corresponding *full* key and value vectors. Calculate the output of the attention block using the top-k keys and values.

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

For in-depth derivation of the steps in SparQ Attention, see Appendix D.

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. Typical ratios are given in Table J2.

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. Note that **Step 3** remains the same. The full code can be found in Appendix B.

3 Experiments and Results

We compare SparQ Attention against two alternative sparse attention methods: H_2O (Zhang et al., 2023), an eviction scheme which iteratively removes tokens from the KV cache that are predicted to not influence future token generation, *FlexGen*, in which the exact top-k highest attention scores are calculated before transferring the corresponding columns of $\boldsymbol{V}$, and *LM-Infinite* (Han et al., 2023), which only transfers a fixed number of initial tokens and recent tokens from the sequence.

We evaluate these different methods in terms of the trade-off between attention memory transfers and accuracy, over five different NLP tasks, with datasets adapted to generate sequence lengths between 1k-2k tokens (see Appendix E). Our experiments span six models: Llama 2 $\{$7B, 13B$\}$, Mistral 7B, and Pythia $\{1.4\text{B}, 2.8\text{B}, 6.9\text{B}\}$ (Touvron et al., 2023; Jiang et al., 2023; Biderman et al.,

Table 1: Results for the largest models tested are presented below. SQuAD and TriviaQA measure performance in accuracy. CNN/DailyMail uses ROUGE-L score. Repetition counts the number of characters before the generation diverges and WikiText task measures perplexity in bits per character (BPC). Values in bold represent the best score for a model, task and sparsity setting. Median standard errors across all models and sparsity settings are: SQuAD 0.7, TriviaQA 0.7, CNN/DailyMail 0.4, WikiText 0.006, 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	32.4	78.7	73.4	69.0	22.1	16.8	15.1	0.61	0.64	0.69	229	76	29
	H_2O		73.2	64.1		78.5	**78.4**		22.2	20.8		**0.61**	**0.63**		61	26
	SparQ		**80.7**	**78.0**		**78.8**	78.2		**22.5**	**22.2**		**0.61**	0.64		**227**	**190**
Mistral 7B	LM-∞	81.0	51.0	31.6	80.9	75.8	72.8	23.7	18.0	16.8	0.62	0.65	0.70	231	81	20
	H_2O		71.2	59.2		**80.8**	**80.6**		**23.5**	**23.4**		**0.63**	**0.65**		38	14
	SparQ		**80.9**	**77.5**		**80.8**	79.0		**23.5**	23.0		**0.63**	**0.65**		**209**	**201**
Pythia 6.9B	LM-∞	57.8	38.5	18.9	52.6	41.6	32.0	20.2	14.9	14.1	0.68	0.71	0.77	150	64	18
	H_2O		52.9	46.6		**52.6**	**52.3**		20.3	18.9		0.69	0.71		47	19
	SparQ		**58.0**	**57.1**		52.4	51.7		**20.6**	**20.6**		**0.68**	**0.70**		**151**	**144**

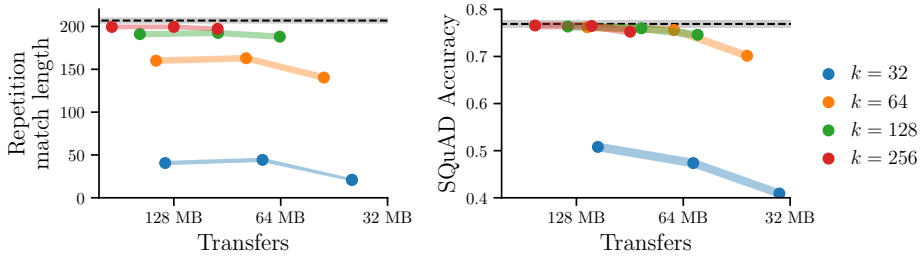


Figure 3: Sweep of hyperparameter configurations ($r \in \{16, 32, 64\}, k \in \{32, 64, 128, 256\}$), for Repetition and SQuAD tasks with Llama 2 7B. Across different models and tasks, we find that setting $k = 128$ and tuning r to achieve the desired trade-off between compression and performance is a robust recipe for selecting hyperparameters.

2023). Results from Llama 2 13B and Mistral 7B are presented in Table 1, with further results in Figures 2, A1 and A2, and show that:

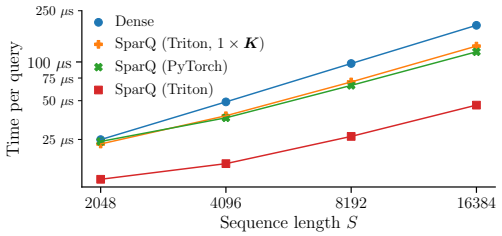
- SparQ Attention performance is robust across all tasks and model sizes tested. Compression ratios of $2\times$ to $8\times$ are readily achievable with little to no loss in task performance.
- The simple recipe of setting $k = 128$ and tuning r to set the compression/performance trade-off seems generally robust over all models and tasks considered (see Figure 3).
- Certain tasks are more challenging for H_2O (Repetition, SQuAD), while others are more forgiving (TriviaQA, WikiText-103).
- LM-Infinite degrades performance across all tasks, demonstrating that the tasks do not permit the trivial solution of discarding the long input sequence.

Additional experiments ablating over different components and aspects of SparQ Attention can be found in Appendix H.[1]

4 Benchmarking

The results above use a theoretical cost model of total memory transfers, allowing us to evaluate SparQ Attention independently of a specific hardware setup. To validate this approach, we performed a set of microbenchmarks of an attention operation in isolation.

[1]Evaluation code is available at `https://github.com/graphcore-research/llm-inference-research/tree/2024-01-paper`.



(a)

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

(b)

Figure 4: (a) Microbenchmark results for A100 (40GB), with batch size 64, 32 heads, $d_h = 128$, $r = 32$, $k = 128$. (b) GPU performance comparison with same hyperparameters and sequence length $S = 4096$.

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

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 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 $\boldsymbol{K}$ twice and using fused gather-then-matmul kernels written using Triton (Tillet et al., 2019), **SparQ (PyTorch)**, with no Triton and **SparQ (Triton,** $1 \times \boldsymbol{K}$**)**, storing $\boldsymbol{K}$ 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.

Our achieved GPU speed-ups are presented in Figure 4, showing strong improvements over the baseline when using the optimisations described above. Standard error for all results given is $< 1\%$ of the mean. See Appendix I 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 size, and to investigate alternatives to storing $\boldsymbol{K}$ twice.

5 Discussion

Sparse attention as an architectural change has been explored by Child et al. (2019); Ren et al. (2021); Beltagy et al. (2020); Zaheer et al. (2020); Kitaev et al. (2020); Tay et al. (2020); Xiao et al. (2023). Compared with existing post-training techniques (Sheng et al., 2023; Mao et al., 2023; Chen et al., 2021; Zhang et al., 2023; Liu et al., 2023), SparQ Attention relies on component-wise sparsity of $\boldsymbol{q}$ and $\boldsymbol{K}$, and introduces $\boldsymbol{V}$ reallocation.

In this work, we explore the scalability of attention mechanisms in modern language models, and find that in many realistic settings, transferring the KV cache from memory creates an LLM inference bottleneck. By analysing the statistics of tensors within attention, we have identified opportunities to approximate attention by sparsely accessing the KV cache, reducing total data transfer.

These opportunities have been realised in SparQ Attention, a novel technique for unlocking faster inference for pre-trained LLMs, without any fine-tuning or modifications to the weights of the model. Our proposed technique modifies the attention mechanism by predicting keys that yield large attention scores, permitting only the relevant tokens from the KV cache to be transferred on every generation step. This allows for pre-trained models to be executed more efficiently.

ACKNOWLEDGEMENTS

We would like to thank Daniel Justus, Paul Balança and Andrew Fitzgibbon for their helpful input and feedback on this work.

REFERENCES

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

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle OBrien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, 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.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34: 17413–17426, 2021.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

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

Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. LM-infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137*, 2023.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

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

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

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

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

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

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *arXiv preprint arXiv:2305.17118*, 2023.

Yuzhen Mao, Martin Ester, and Ke Li. 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.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

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/`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5, 2023.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Hongyu Ren, Hanjun Dai, Zihang Dai, Mengjiao Yang, Jure Leskovec, Dale Schuurmans, and Bo Dai. Combiner: Full attention transformer with sparse computation cost. *Advances in Neural Information Processing Systems*, 34:22470–22482, 2021.

Alexis Roche, Grégoire Malandain, Xavier Pennec, and Nicholas Ayache. The correlation ratio as a new similarity measure for multimodal image registration. In *Medical Image Computing and Computer-Assisted Intervention MICCAI98: First International Conference Cambridge, MA, USA, October 11–13, 1998 Proceedings 1*, pp. 1115–1124. Springer, 1998.

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

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

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: high-throughput generative inference of large language models with a single GPU. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. In *International Conference on Machine Learning*, pp. 9438–9447. PMLR, 2020.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. 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.

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

Jesse Vig. 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.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022.

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

Chulhee Yun, Yin-Wen Chang, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. $O(n)$ connections are expressive enough: Universal approximability of sparse transformers. *Advances in Neural Information Processing Systems*, 33:13783–13794, 2020.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297, 2020.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.

A Detailed Results

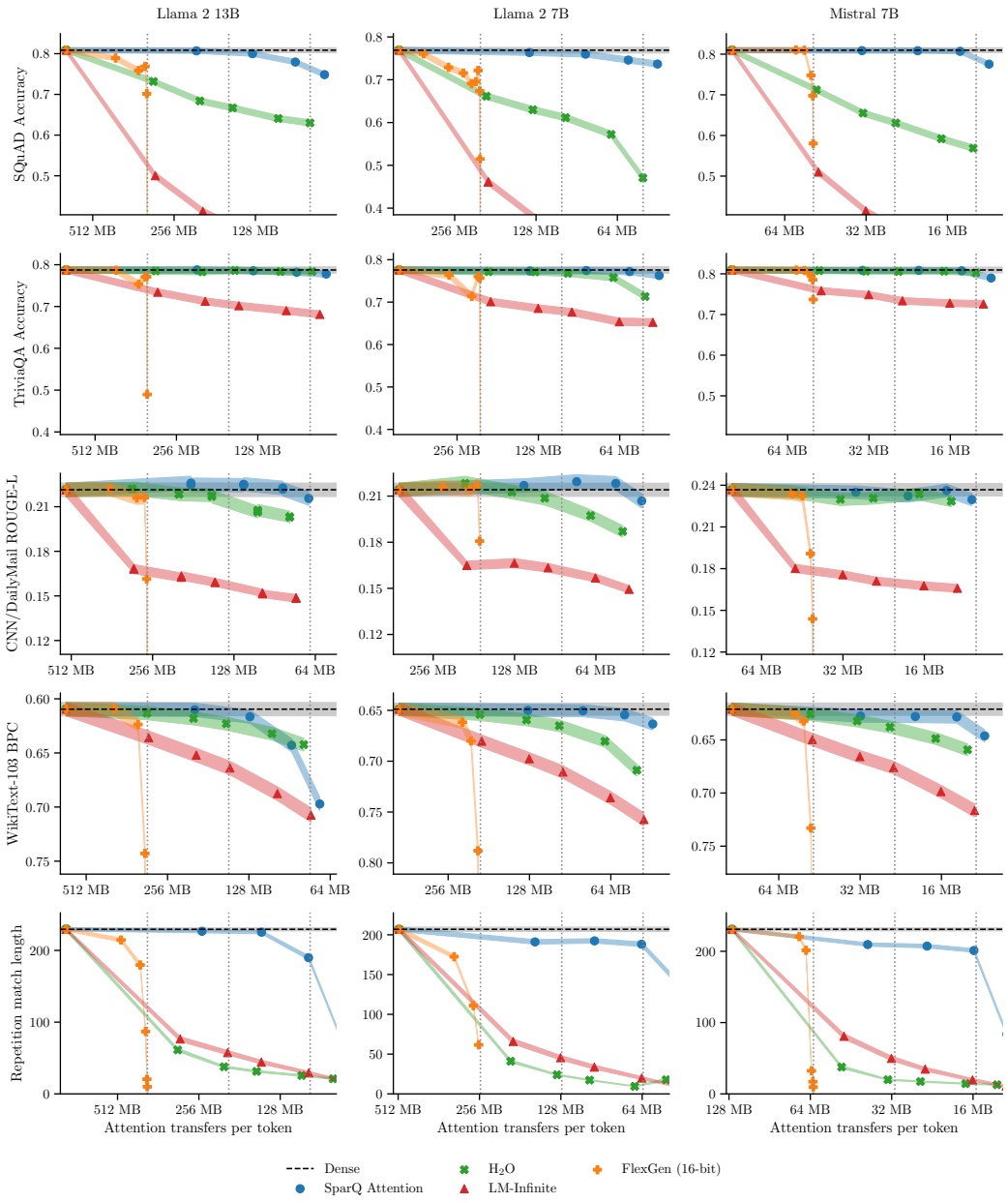


Figure A1: Compression versus performance trade-off curves over all tasks and multiple models. The y-axis minimum is set to (0.5, 0.5, 0.5, 1.25, 0.0)× the dense baseline for the tasks, reading top-to-bottom, in order to give a consistent view of the performance loss across models. Vertical dotted lines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

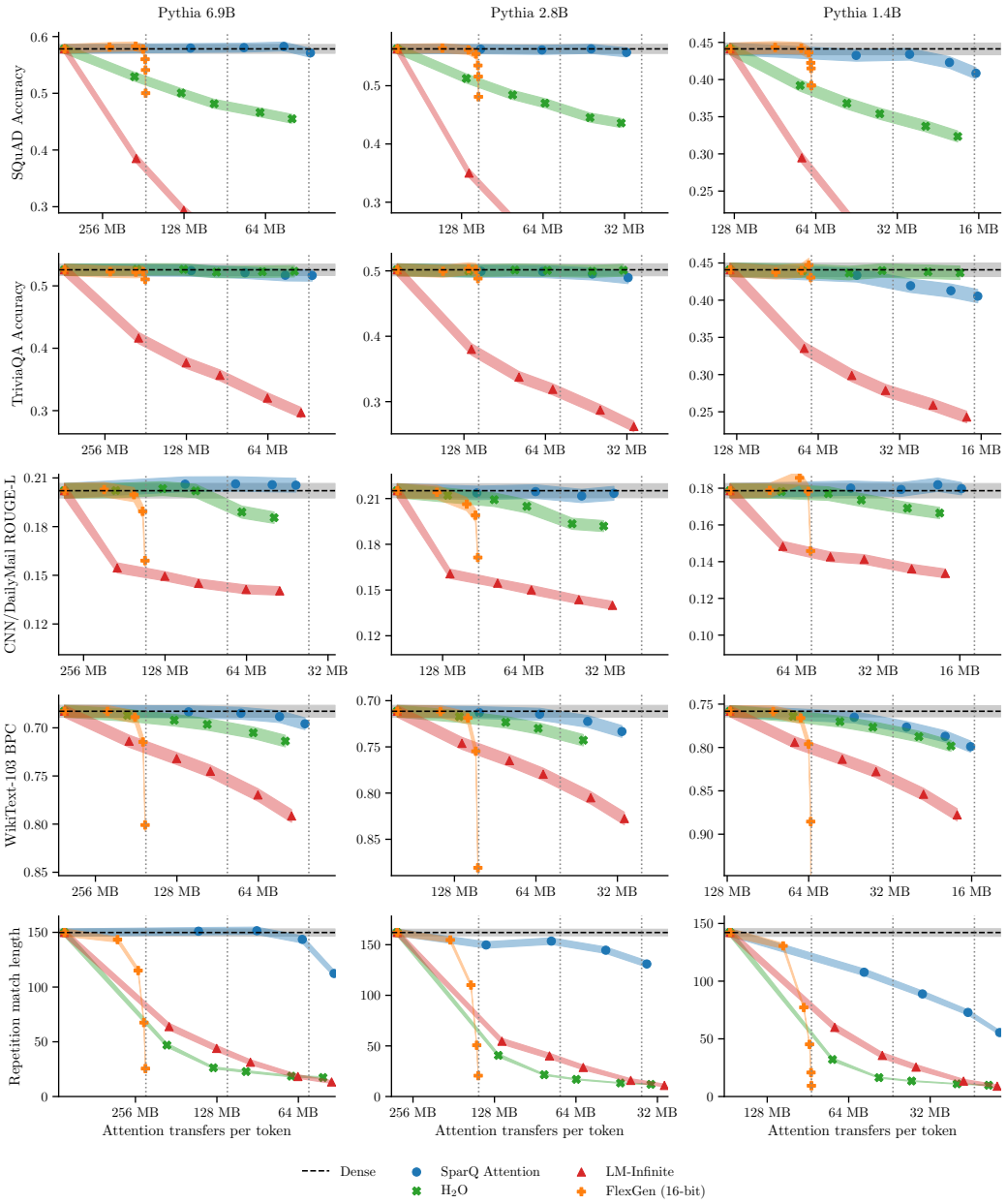


Figure A2: Compression versus performance trade-off curves for Pythia models (see Figure A1).

B CODE

```
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 ATTENTION MEMORY TRANSFERS

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

$$\mathcal{M}_{\text{base}} = \underbrace{2\,S\,d_h}_{\text{Read } \boldsymbol{K} \text{ and } \boldsymbol{V}} + \underbrace{2\,d_h}_{\text{Write current } \boldsymbol{k} \text{ and } \boldsymbol{v}} \tag{C1}$$

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

$$\mathcal{M}_{\text{SparQ}} = \underbrace{S\,r}_{\text{Read } r \text{ rows of } \boldsymbol{K}} + \underbrace{2\,k\,d_h}_{\text{Read top-}k \text{ columns of } \boldsymbol{K} \text{ and } \boldsymbol{V}} + \underbrace{4\,d_h}_{\text{Write current } \boldsymbol{k} \text{ and } \boldsymbol{v}, \text{ read/write } \overline{\boldsymbol{v}}} \tag{C2}$$

D SPARQ ATTENTION PROCEDURE

The SparQ AttentionAttention procedure consists of three steps. In the first step we find the indices $\boldsymbol{i}_1 \in \mathbb{N}^r$ corresponding to the top $r \in \mathbb{N}$ components of the query vector $\boldsymbol{q} \in \mathbb{R}^{d_h}$ and proceed to calculate the *approximate* attention scores $\hat{\boldsymbol{s}} \in \mathbb{R}^S$ across the sequence as follows, where $S \in \mathbb{N}$ is the sequence length:

$$\begin{aligned} \hat{a}_i &= \sum_{j \in \boldsymbol{i}_1} q_j K_{i,j} \\ \hat{\boldsymbol{s}} &= \sigma\left(\frac{\hat{\boldsymbol{a}}}{\tau}\right) \end{aligned} \tag{D1}$$

where $\sigma(\boldsymbol{x})_i = e^{x_i} / \sum_j (e^{x_j})$ is the $\mathrm{softmax}$ function and $\boldsymbol{K} \in \mathbb{R}^{S \times d_h}$ is the key cache matrix. In the case of standard attention, the scaling factor τ is chosen to be $\sqrt{d_h}$, corresponding to the length of the vector $\boldsymbol{q}$ over which the dot product summation is performed. However, since the dot product is approximated using only r components here, the scaling factor needs to be appropriately changed. Empirically we found the appropriate factor to be:

$$\tau = \sqrt{d_h \frac{\|\boldsymbol{q}_{[\boldsymbol{i}_2]}\|_1}{\|\boldsymbol{q}\|_1}} \tag{D2}$$

which takes into account the relative L1-norm of the selected top r components of the vector $\boldsymbol{q}$.

In the second step, we proceed to find indices $\boldsymbol{i}_2 \in \mathbb{N}^k$ corresponding to the top $k \in \mathbb{N}$ components of the approximate score vector $\hat{\boldsymbol{s}}$. We then calculate the attention scores using only the top-k positions, which we can express by defining a corresponding boolean mask $\boldsymbol{b} \in \{0,1\}^S$:

$$b_i = \begin{cases} 1 & \text{if } i \in \boldsymbol{i}_2 \\ 0 & \text{else} \end{cases}$$
$$\boldsymbol{s} = \sigma\Big(\frac{(\boldsymbol{q} \cdot \boldsymbol{K}^\top) \circ \boldsymbol{b}}{\sqrt{d_h}}\Big) \tag{D3}$$

In the third step, we perform a weighted sum of the value vectors in $V \in \mathbb{R}^{S \times d_h}$ using the calculated scores $\boldsymbol{s}$. In order to improve the accuracy of the final approximation, we add an additional weighted contribution of the mean value vector $\bar{\boldsymbol{v}} = \frac{1}{S} \sum_i \boldsymbol{V}_{i,*}$:

$$\boldsymbol{y} = \alpha \sum_{i \in \boldsymbol{i}_2} s_i \boldsymbol{V}_{i,*} + (1 - \alpha)\bar{\boldsymbol{v}} \tag{D4}$$

where $\alpha \in [0,1]$ is the relative weight of the top-k terms. We choose α using the approximate attention scores $\hat{\boldsymbol{s}}$ from the first step as:

$$\alpha = \sum_{i \in \boldsymbol{i}_2} \hat{s}_i \tag{D5}$$

E Tasks and Datasets

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 between 1k and 2k tokens. As we wanted 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. We quote performance for sub-word language modelling in BPC, to account for any differences in vocabulary across models.

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 required 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.

F ARITHMETIC INTENSITY

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 A compute unit capable of $r_{\mathcal{A}}$ scalar arithmetic operations per second is connected to a memory via an interface that can transfer $r_{\mathcal{M}}$ scalar elements per second, processing a workload requiring $\mathcal{A}$ arithmetic operations and $\mathcal{M}$ transfers. Assuming concurrent compute and data transfer, when the *arithmetic intensity* $\mathcal{A}/\mathcal{M}$ of the workload is less than the ratio $r_{\mathcal{A}}/r_{\mathcal{M}}$, execution time is limited by $r_{\mathcal{M}}$.

Sequence generation Consider a full transformer layer, with N parameters, batch size B, and C elements in the attention KV cache per batch element. We assume Grouped Query Attention (GQA) (Ainslie et al., 2023) with g grouped-query heads ($g = 1$ for standard multi-head attention). This implies the arithmetic intensity:

$$\frac{\mathcal{A}}{\mathcal{M}} = \frac{BN + BCg}{N + BC} = \frac{N + Cg}{N/B + C} \tag{F1}$$

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. An alternative formulation for a standard transformer with model dimension d_m and sequence-length S, has $N = 12(d_m)^2$ and $C = 2\,S\,d_m/g$, giving:

$$\frac{\mathcal{A}}{\mathcal{M}} = \frac{6 + \rho\, g}{6/B + \rho} \tag{F2}$$

where $\rho = S/(gd_m)$. The value ρ underlies the KV cache-model size relationship outlined above, determining the point at which the model becomes memory bandwidth bound (Figure F1). Since long sequences are desirable, data transfer is the performance-limiting factor, motivating the search for transfer-efficient approximations to full attention.

Following this framework, we provide concrete examples of arithmetic intensity for various models and the implications for execution modern machine learning hardware. We observe from Equation (F2) that the arithmetic intensity as batch size increases approaches $g + 6/\rho$. For example:

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

Hardware Properties of selected machine learning hardware.[2] 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.

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

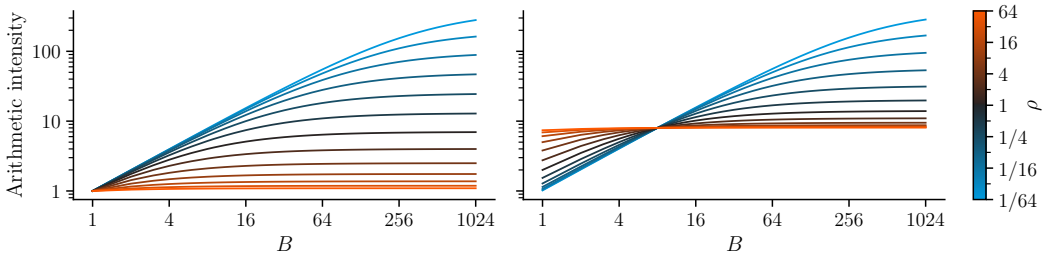


Figure F1: Relationship between $\rho = S/(gd_m)$, batch size B and arithmetic intensity during sequence generation. *Left:* Multi-head attention. *Right:* GQA ($g = 8$). ML hardware provides $r_{\mathcal{A}}/r_{\mathcal{M}} > 200$, making memory bandwidth the limiting factor in many practical scenarios (see Appendix F).

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.

G Attention Sparsity Analysis

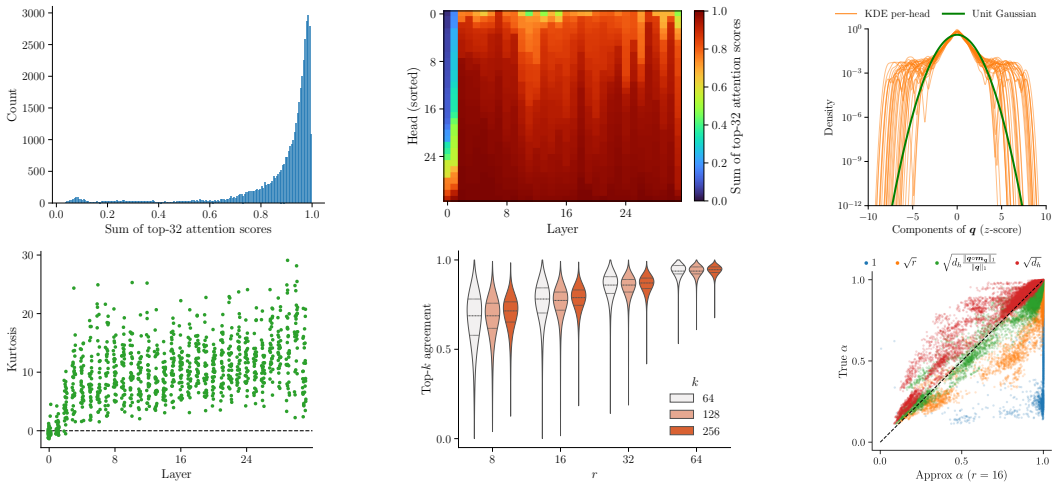


Figure G1: Statistics of Llama 2 7B over 40 SQuAD queries, for all 32 layers × 32 heads unless noted. (*Top Left*) Sum softmax output allocated to the 32 highest-scoring positions, demonstrating natural attention sparsity; (*Top Middle*) for each head. (*Top Right*) Kernel density estimate (Rosenblatt, 1956) of components of $\boldsymbol{q}$ in layer 16, showing heavy tails. (*Bottom Left*) Fisher Kurtosis of $\boldsymbol{q}$ components, showing that the query vector is leptokurtic for most heads. (*Bottom Middle*) Top-k agreement, the proportion of the top-k positions that are correctly predicted by an approximated softmax for various r and k. (*Bottom Right*) 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 temperature.

In order to understand how to approximate attention in pre-trained transformers, we analysed the queries, values and intermediate *scores* vector (softmax output). We took 40 examples from our

Table G1: *Excess* correlation ratio η (Roche et al., 1998) along axes of $\boldsymbol{V}$ (excess: subtract $d^{-0.5}$, so uniform random data $= 0.0$). This demonstrates substantial auto-correlation along the sequence axis. Calculated for Llama 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

SQuAD 1-shot task, and generated the first completion token using the dense Llama 2 7B model, capturing the $\boldsymbol{q}$ vector and $\boldsymbol{K}$, $\boldsymbol{V}$ matrices from every layer and attention head, showing derived statistics in Figures G1 to G3 and Table G1.

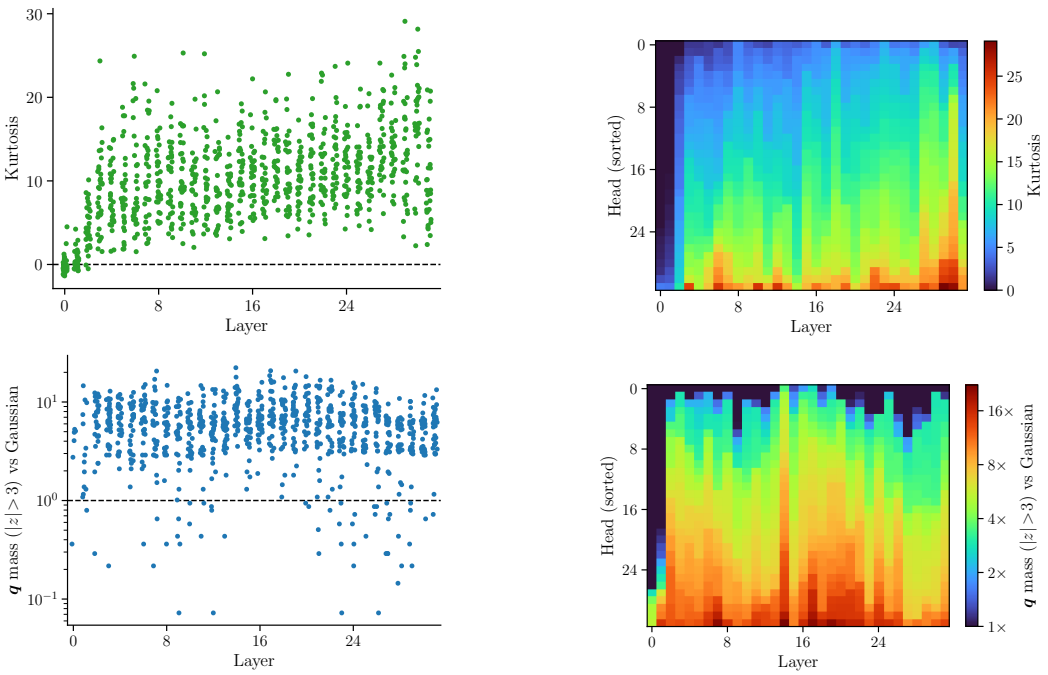


Figure G2: Statistics of components of $\boldsymbol{q}$ for each head, as a function of layer. (Top) Kurtosis (Fisher), indicating that most heads have heavy-tailed $\boldsymbol{q}$. (Bottom) z-value mass, normalised by that of a Gaussian (0.3%), showing that most heads are outlier-heavy. All Llama 2 7B, measured over 40 SQuAD examples.

In Figure G1 (*Top Right*) 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× 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 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 G3 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

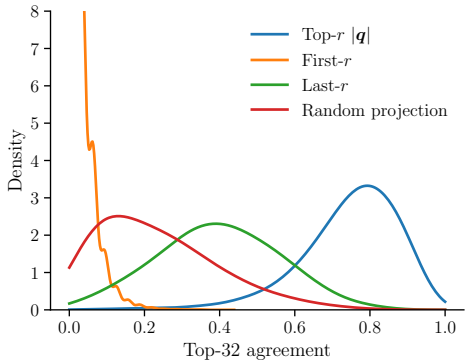


Figure G3: Top-k agreement between approximate and true scores (Llama 2 7B, measured over 40 SQuAD examples). Top-k agreement is the proportion of the top-k positions that are correctly predicted by an approximated softmax, using a projection of $\boldsymbol{q}$, either component-wise or a random low-rank projection.

sparsity of 50% and 75% maintain high top-k correspondence to the exact attention scores, which is consistently maintained over various values of k.

It is useful to drop positions in $\boldsymbol{V}$ given attention scores, but this can save at most half of the data transfers, since the whole of $\boldsymbol{K}$ is needed to calculate these scores. We propose approximating these scores using a subset of the components of $\boldsymbol{K}$. 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 $\boldsymbol{K}$ matrix, instead only transferring some components of $\boldsymbol{K}$ for all positions, then all components of $\boldsymbol{K}$ for some positions.

Our hypothesis is that the r largest-magnitude components of $\boldsymbol{q}$ are most useful to predicting the score, $\boldsymbol{q}\boldsymbol{K}^\top$. The coverage of this technique against an arbitrary-component baseline is shown in Figure G3. 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.

H 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 any data transfer. The results in Figure H1 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 G1 (*Bottom Right*), 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 our own proposed temperature, defined in Equation (D2). 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 H2.

Hyperparameter selection The reduction of data transfer attained by SparQ Attention is controlled by its two hyperparameters, k and r. Reducing either of these variables will improve the bandwidth efficiency, but can negatively impact task performance. Figure 3 shows the relationship between k and r on both of these factors. Based on these results, we propose a simple recipe of set-

ting $k = 128$ and tuning r to maintain a good trade-off between data transfer and task performance for a range of models and tasks.

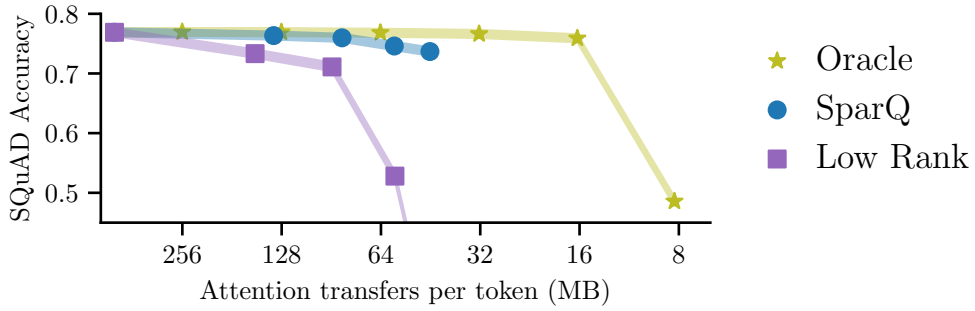


Figure H1: SQuAD 1-shot accuracy with Llama 2 7B of SparQ Attention and a random low rank compression scheme against an oracle top-k selector.

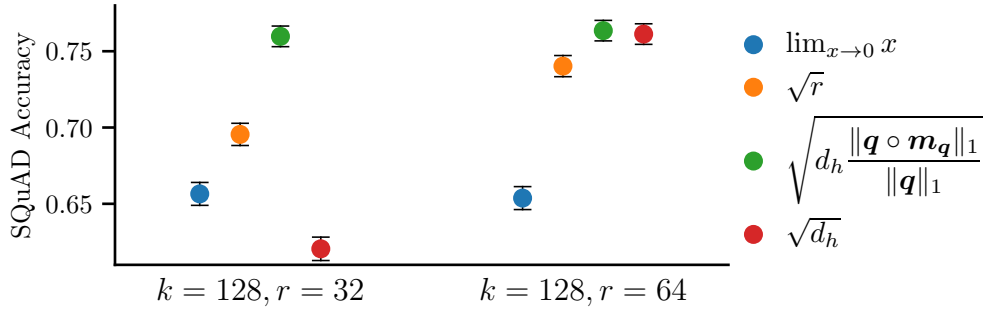


Figure H2: Comparison of different softmax temperatures for approximate attention scores for two different hyperparameter configurations (Llama 2 7B SQuAD 1-shot performance).

I BENCHMARKING DETAIL

Benchmarking code is made available at
`https://github.com/graphcore-research/llm-inference-research/tree/2024-01-paper`.

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 4 and Figure 4, we give an aggregate picture of the trends in Figure I1. 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

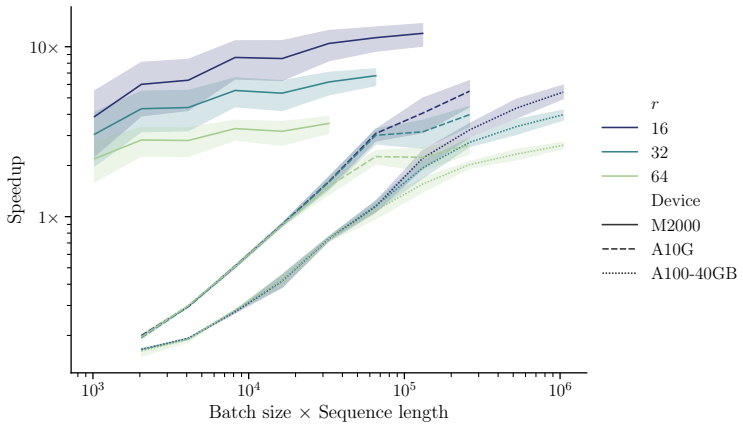


Figure I1: SparQ speedup over the dense baseline, across a range of batch size (1-64), sequence length (1024-65536) and k (64-512), for different devices. We note that for both GPUs, the number of KV elements is a limiting factor for the achieved speedup, and that this could be improved by writing a fully fused SparQ Attention kernel.

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.

Storing $\boldsymbol{K}$ twice One limitation of a theoretical model of data transfer is that it does not account for the granularity of memory access. Since the $\boldsymbol{K}$ 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 $\boldsymbol{K}$ 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 $\boldsymbol{k}$ twice. This extra write is non-contiguous, but small, so should not form a bottleneck.

J METHODOLOGY

We provide a comprehensive set of hyperparameters for reference in Table J1. Typical compression ratios for settings of (r, k) are given in Table J2.

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 ratio 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.

Group	Hyperparameter	Value or range
Dense model	Family	Llama 2 (13B, 7B), Mistral (7B), Pythia (6.9B, 2.8B, 1.4B)
	d_h	$\{80, 128\}$
	Max S	$\{2048, 4096\}$
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	Eviction	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 $k-16$ $k \in \{192, 256, 384, 512, 768\}$
SparQ Attention	Rank r	$\{16, 32, 64\}$
	Number of values k	128
	Local window l	$k/4$

Table J1: Experiment hyperparameters

Method	k	r	Compression
SparQ Attention	128	16	0.13 - 0.17
		32	0.19 - 0.23
		64	0.31 - 0.36
H_2O	192	–	0.10 - 0.17
	256		0.13 - 0.22
	384		0.20 - 0.33
	512		0.26 - 0.43
	768		0.39 - 0.65

Table J2: Range of compression ratios for different settings of (r, k), for Llama 2 7B and Pythia 6.9B. The compression ratio achieved varies across models and tasks, based on the sequence length and head size.

J.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.

J.1.1 QUESTION ANSWERING (SQUAD 1-SHOT)

```
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
```

J.1.2 QUESTION ANSWERING (TRIVIAQA 0-SHOT)

```
Apritifs and digestifs ( and) are drinks, typical...
Apritifs
An apritif is an alcoholic beverage usually serve...
"Apritif" may also refer to a snack that precedes...
"Apritif" 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).

J.1.3 SUMMARISATION (CNN/DAILYMAIL)

```
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...
```

J.1.4 REPETITION (SHAKESPEARE)

```
 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.
```

J.1.5 LANGUAGE MODELLING (WIKITEXT-103)

```
= 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
```