Analyzing Inference Optimizations for Transformers

Rishab Parthasarathy and Reece Shuttleworth

***Abstract*—The transformer architecture is the backbone of Large Language Models. These models are notorious for the computational intensity required to run them and therefore also their significant energy usage. As LLM-based AI products become more prominent, the inference of this architecture will need to become more efficient.**

In this work, we study two inference optimizations that can reduce the energy and computation required during inference. First, we analyze a performance optimization called KV Caching, which reduces the amount of repeated computation during inference by storing the keys and values from previous inference passes. Second, we analyze a model design choice underlying the differences between Multi-Head Attention, Grouped-Query Attention, and Multi-Query Attention, namely changing the number of groups of keys and values that are used in attention.

We show that energy usage, cycles, and compute scale linearly with the number of query groups selected across a variety of different head counts. Noticeably, when measuring the compute with a model with 16 heads, we notice a more than 15% drop in computes when going from Multi-Head Attention to Multi-Query Attention. We also show that increasing the size of the KV cache leads to noticeable improvements in the compute and energy usage of the model due to the reduction of redundant computation.

I. Introduction

Investment and usage of AI products has seen a huge increase in recent years due to the development of the machine learning architecture called the Transformer [15]. This architecture is the backbone of Large Language Models like GPT-4, whose surge in popularity has not abated. Much of the success of these models can be attributed to scaling: making the models larger and training them on more and more data have been the two most important things influencing the increasing performance of these models [13]. Because of this, these models are very expensive to serve for inference. Some estimates state that OpenAI incurs expenses of over $700,000 per day to serve ChatGPT [6]. To combat this, some models have sacrificed using the optimal tradeoff between model size and training data size [8] by training smaller models, like LLaMA [14], on huge datasets, to reduce the computational intensity of inference. However, these models are still billions if not tens of billions of parameters, meaning they are still difficult to serve, especially on consumer hardware.

Serving these models efficiently during inference is crucial, especially since their current adaption is straining existing technological infrastructure. Improving the efficiency of inference can help to alleviate this strain and reduce the energy consumption of these power-hungry models. Improving their efficiency can also help to democratize transformers by enabling consumer hardware to power them. In this work, we investigate the impact of several interventions on the efficiency of inference.

The first method we investigate is the effect of using KV caching during inference to reduce the amount of redundant computation performed. The second method we investigate is the effect of using different query group sizes on computation and energy, since using fewer query groups results in smaller matrix multiplies and instead enables reuse of keys and values across attention heads.

II. Background

In this section, we provide a technical background to the topics we investigate.

A. Attention

In transformers, their attention module is the differentiating factor that makes them so powerful [15]. It takes as input a vector for every token in the sequence, each of which is the length of the models hidden dimension. First, this collection of vectors is projected three different times to get a collection of queries, keys, and values. Then, attention takes the dot product of every query with every key and scales these values. By then softmaxing these values with respect to the query dimension, we get an attention map describing how much each token proportionally 'attends' to every other token. We then do the dot product with the values and this output attention map to get the final result. We do this process separately for each head in the model.

Attention for one head is described by the equation below, where X is the input of size sequence length by hidden dimension.

$$attention(X) = softmax(\frac{(XW_Q)(W_K^T X^T)}{\sqrt{d_{hidden}}})(XW_V)$$

As seen in the above equation, the main cost of attention comes from the 6 matrix multiplications that must be performed. In the subsequent sections of this paper, we address a number of interventions to reduce the costs of these matrix multiplications.

B. KV Caching

KV caching [11] is an inference optimization that exploits the fact that transformers reuse the same results when doing multiple passes through the model during auto-regressive text generation. This reuse is due to the fact that previous tokens reuse their key and value representations and have the same unnormalized attention score across inference passes. Since we must do an inference pass for every single new token we generate, a naive implementation frequently recomputes many queries, keys and values. To reiterate, when auto-regressively generating text, only the newly generated token has queries, keys, and values that have not been previously generated. This fact can be exploited by caching the key and value representations and storing the unnormalized attention scores rather than recomputing them for each forward pass. Ideally, when trying to predict each next token during auto-regressive text generation, we would store all previously computed vectors. However, storing all these vectors may be unfeasible due to memory constraints.

It is still possible to partially exploit this reuse without using as much memory by instead storing just the last n keys and values of the sequence. For sequences of less then length n, this acts as full KV caching. For sequences longer than n, we partially recompute keys and values and reuse the vectors that were stored.

C. Grouped-Query Attention

The original paper that introduced the transformer introduced Multi-Head Attention (MHA) [15], meaning that every attention head operated on a distinct set of queries, keys, and values. However, recent models like LLaMA-2 [14] have instead adapted Grouped-Query Attention (GQA) [1], which shares sets of keys and values across several sets of queries. Some models, like Falcon [2], push this even further and use Multi-Query Attention (MQA) [12], which only use one set of keys and values and share them across all sets of queries. This spectrum of versions of attention can be thought of as changing one specific variable: the number of unique sets of keys and values generated.

Generating fewer unique sets of keys and values across a constant number of attention heads has clear benefits due to the reduced size of matrix multiplies performed and the ability to instead reuse the keys and values across attention heads. More importantly, previous work has shown that reducing the number of keys and values did not noticeably effect performance and was therefore effective at reducing the parameter count without hurting model performance [1].

D. GPT-J

GPT-J is a six billion parameter decoder-only language model that was trained by EleutherAI in 2021. Inspired by GPT-3, GPT-J was trained on the Pile for general language learning capabilities [16]. We utilize GPT-J because it is a standard across interpretability studies, with a standard transformer backbone, which we subsequently augment with multi-query and grouped-query attention. Specifically, we utilize GPT-J with 16 layers rather than the full 28 due to the memory limitations of our machines.

E. Eyeriss

Eyeriss [4] is a state-of-the-art spatial neural network accelerator is optimized for energy efficency by utilizing a row-stationary dataflow. For our experiments, we model GPT-J running on an Eyeriss-like accelerator that is supported in Accelergy and Timeloop.

F. Accelergy/Timeloop

For our energy and cycle evaluations, we utilize Accelergy and Timeloop, state of the art methods for optimizing loop mappings and approximating the number of cycles/energy needed. Specifically, we utilize Timeloop to search the space of optimal loop mappings, and we use Accelergy to evaluate these mappings on the target hardware and generate energy usage estimations [10], [17].

III. Prior Work

The existing work on KV-Caches has focused on profiling both mean FLOP utilization (MFU) and the latency per token, where MFU serves as a proxy metric for both throughput and arithmetic intensity [11]. We note that this focus largely comes because companies serving large models are focused on improving their serving capacity, leading them to focus on throughput and latency.

We propose to take this approach and instead address it from the context of Eyeriss, which is more focused on energy-constraints. Specifically, we plan to explore how different KV-cache sizes interact with general spatial accelerator workflows, which are only profiled at a surface level in the current work.

Beyond the basic KV-cache evaluation that we plan, existing work has targeted studying the matrix multiplication operations in Transformers using Timeloop, comparing MQA and the basic multi-head attention [9]. However, these works use the Berkeley Gemmini architecture [7] for generalizable systolic array architectures, which are similar in practice to Tensor Processing Units (TPUs).

We instead propose profiling these architectures and dataflows on general spatial accelerators like Eyeriss, which prioritizes energy efficiency and thus aligns with the goals of our study. We also propose profiling a more recent implementation of attention for inference, Grouped Query Attention, which aims to further improve memory footprint by further grouping queries from MQA [1].

IV. Analyzing KV Cache

In this experiment, we aim to investigate the effect of the number of keys and values stored in the KV cache on the energy consumption and number of cycles and computes performed during inference through the GPT-J [16] transformer model. We use Accelergy [17] and Timeloop to measure the models performance and use the Eyeriss [4] architecture as our architecture accelerator.

For our experiment, we assume a sequence length of 512 tokens and sweep KV cache size in $\{0, 1, 4, 16, 64, 256, 512\}$. We report Energy usage, cycles, and computes in Figure 1. We observe that the computations performed drops linearly as the number of keys and values cached increases. Cycles and Energy both also drop consistently as higher numbers of keys and values are cached for inference.

In Table I, We look closer at the percent drop of our statistics in relation to doing no caching. We observe that when we put all 512 tokens in the cache, which can be thought of as putting all tokens in our sequence length in the cache, we

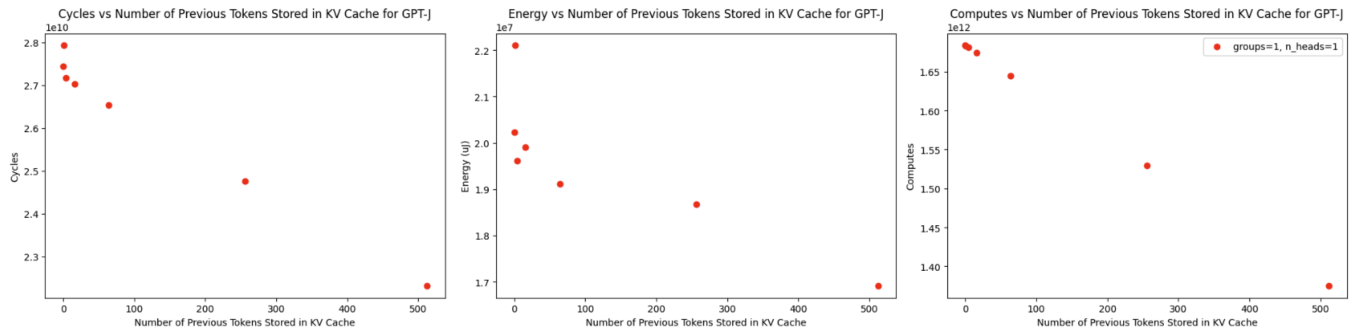

Fig. 1. The effect of storing more keys and values in the KV cache on the number of cycles, amount of energy used, and number of computes. This experiment assumes a sequence length of 512 and uses the GPT-J architecture. We notice that __

# of tokens in cache	Energy	Cycles	Computes
0	0%	0%	0%
1	-9.29%	-1.77%	0.04%
4	3.02%	1.00%	0.14%
16	1.59%	1.53%	0.57%
64	5.48%	3.29%	2.30%
256	7.68%	9.80%	9.18%
512	16.36%	18.68%	18.33%

TABLE I
PERCENT REDUCTION IN COMPARISON TO NOT USING KV CACHE. EXPERIMENT USES A SEQUENCE LENGTH OF 512.

have a 16.36% reduction in energy usage, 18.33% reduction in computes, and 18.68% reduction in cycles performed. This shows that notable improvements to the efficiency of transformer inference can be achieved by using a KV cache when possible.

We note that the increase in computation from small cache sizes like 1 likely results from odd numbers and non-powers of two being introduced into the computation, resulting in increased cycles and energy usage. We note that the larger KV cache sizes still produce a linear relationship, making the small values outliers.

V. ANALYZING QUERY GROUP SIZE

In this experiment, we aim to investigate the effect of changing the number of query groups used on the energy consumption and number of cycles and computes required to perform inference for the GPT-J [16] transformer model. To do this, we model the GPT-J architecture in Accelergy [17] and Timeloop and use the Eyeriss [4] architecture as our accelerator, which is supported by the Accelergy and Timeloop libraries.

We use two variables to modify the attention mechanism: while keeping the hidden dimension fixed, we modify both the number of heads used in the attention mechanism as well as the number of query groups used. We sweep number of heads in $\{16, 64, 256\}$ and number of query groups in $\{1, 2, 4, 8, 16\}$ and for each combination of these two variables simulate their performance statistics. We report Energy Usage, Cycles, and Computes in Figure 2. We observe that energy and computes scale linearly with the number of query groups, meaning that reducing the number of query groups results in less energy usage and fewer computations performed.

Query Groups	Energy	Cycles	Computes
1	13.95%	13.85%	15.31%
2	12.54%	13.587%	14.29%
4	11.05%	7.913%	12.245%
8	8.89%	0%	8.16%
16	0%	0%	0%

TABLE II
PERCENT REDUCTION IN COMPARISON TO MULTI-HEAD ATTENTION (USING 16 GROUPS) FOR THE 16 HEAD EXPERIMENT.

In Table II, we look more closely into the case where number of heads is equal to 16. Looking at this table, we can see the percentage increase achieved for a certain number of query groups in comparison to using 16 query groups, like normal Multi-Head Attention does. We observe that using 1 query group, which is Multi-Query Attention, results in a 13.95% reduction in Energy usage and 15.31% in computation performed.

Interestingly, our results show that for any particular number of query groups, data points with more heads have less computation or energy. This is because since we are fixing the hidden dimension to a constant value, increasing the number of heads implicitly decreases the head dimension, which results in a overall decrease in the number of computations performed. However, in practice this cannot be exploited to increase the efficiently of inference past a certain point because the size of the head dimension must be large enough to contain rich semantic information, and reducing the head dimension past a certain point removes that ability. Investigating this tradeoff is outside the scope of this work.

We note that we do not report accuracy because of the compute requirement to pretrain language models from scratch. Instead, we note that the GQA paper has already studied accuracy over an array of summarization and language modeling tasks, finding that GQA and MQA do not suffer significant accuracy losses over the more expressive MHA [1].

VI. FUTURE WORK

In the future, work could explore modifying the total hidden dimension allocated to each head, and how the resulting

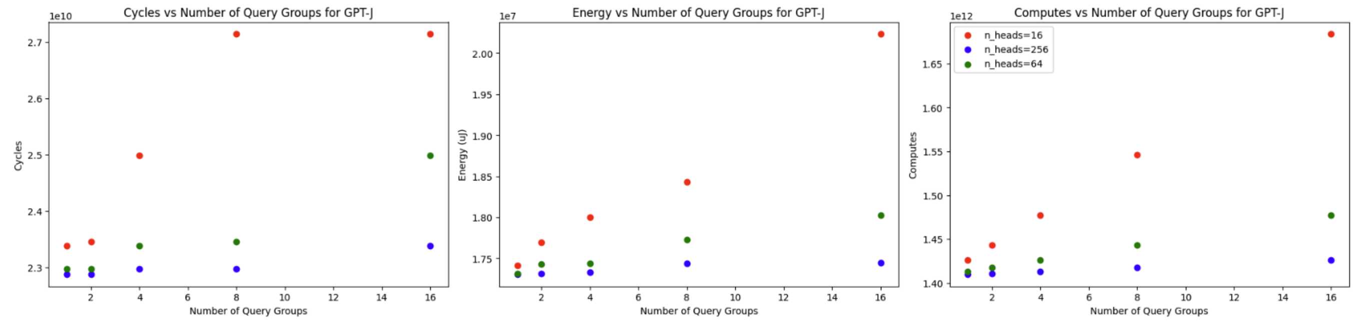

Fig. 2. The effect of increasing the number of query groups (and therefore decreasing the number of keys and values) on the number of cycles, amount of energy used, and number of computes.

changes in head dimension size impact training performance, as discussed in Section V. On top of that, future studies could address more recent advances like speculative decoding [3], where multiple tokens are decoded simultaneously, or kernel-level optimizations like FlashAttention [5]. Each of these new techniques introduces unique, new challenges to profile, with the tradeoff of additional heads introduced in speculative decoding or the fused softmax used by FlashAttention yet to be profiled extensively on energy-efficiency.

VII. Conclusion/Discussion

In this work, we analyzed two interventions that can be used to improve the inference efficiency of transformer models. First, we investigated KV caching, an optimization designed to reduce or eliminate the amount of recomputation in the models attention mechanism. We showed that KV caching can lead to clear reductions in energy usage, computations performed, and cycles used. In particular, we showed that using a full KV cache can reduce the energy consumption of inference by 16.36% in comparison to the no caching baseline.

Second, we investigated how the model architectural change of using fewer query groups in attention can impact the efficiency of inference. We showed that energy usage and computes performed scales linearly with the number of query groups, and that using Multi-Query Attention instead of Multi-Head Attention results in an energy reduction of 13.95%.

Improving the optimization efficiency of transformers will be essential for these models to realize the potential benefits they promise to offer. Right now, inference on these state-of-the-art models requires expensive hardware that uses huge amounts of energy, prohibiting all but a few from running these models. In the short term, inference optimizations can help to increase the efficiency of these models, which would help to both reduce the barrier to entry to using transformer models and also reducing the environmental impact of these models.

More work will be needed to make these methods more useful and to find more novel methods that can be combined to make inference in tranformers more efficient. However, we are optimistic that there are many ways to make these models more efficient and hope that these optimizations can be used to effectively deploy AI products around the world.

VIII. Submission Details

Rishab and Reece equally planned the direction of this project. Rishab worked on the scripts to process GPT-J into the YAML format taken by Timeloop, and he also created the scripts to run Timeloop and Accelergy on the generated experiments. He also contributed to writing the paper. Reece led the writing of the paper and also wrote scripts to process Timeloop/Accelergy outputs into the diagrams/tables in this paper.

The submitted repository can be found at this GitHub repo.

References

[1] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, “GQA: training generalized multi-query transformer models from multi-head checkpoints,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Association for Computational Linguistics, 2023, pp. 4895–4901. [Online]. Available: https://aclanthology.org/2023.emnlp-main.298

[2] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, Étienne Goffinet, D. Hesslow, J. Launay, Q. Malartic, D. Mazzotta, B. Noune, B. Pannier, and G. Penedo, “The falcon series of open language models,” 2023.

[3] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao, “Medusa: Simple LLM inference acceleration framework with multiple decoding heads,” *CoRR*, vol. abs/2401.10774, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.10774

[4] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[5] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, “Flashattention: Fast and memory-efficient exact attention with io-awareness,” in *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html

[6] G. Elimian, “Chatgpt costs $700,000 to run daily, openai may go bankrupt in 2024- report,” *technext24*, 2024.

[7] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, “Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration,” in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE Press, 2021, p. 769–774. [Online]. Available: https://doi.org/10.1109/DAC18074.2021.9586216

[8] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre, "Training compute-optimal large language models," 2022.

[9] S. Kim, C. Hooper, T. Wattanawong, M. Kang, R. Yan, H. Genc, G. Dinh, Q. Huang, K. Keutzer, M. W. Mahoney, Y. S. Shao, and A. Gholami, "Full stack optimization of transformer inference: a survey," *CoRR*, vol. abs/2302.14017, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2302.14017

[10] A. Parashar, P. Raina, Y. S. Shao, Y. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. S. Emer, "Timeloop: A systematic approach to DNN accelerator evaluation," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2019, Madison, WI, USA, March 24-26, 2019*. IEEE, 2019, pp. 304–315. [Online]. Available: https://doi.org/10.1109/ISPASS.2019.00042

[11] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Levskaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," 2022.

[12] N. Shazeer, "Fast transformer decoding: One write-head is all you need," *CoRR*, vol. abs/1911.02150, 2019. [Online]. Available: http://arxiv.org/abs/1911.02150

[13] R. S. Sutton. (2019) The bitter lesson. [Online]. Available: https://www.cs.utexas.edu/~eunsol/courses/data/bitter_lesson.pdf

[14] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," 2023.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[16] B. Wang and A. Komatsuzaki, "GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model," https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

[17] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2019.