Reproduction attempt of E.T.: re-thinking self-attention for transformer models on GPUs paper

Krishna Panthi
School of Computing, Clemson University
Clemson, South Carolina, USA
Email: kpanthi@clemson.edu

***Abstract*—Transformer models have become ubiquitous in natural language processing (NLP) and are increasingly being used in other domains like computer vision. However, their large size and computational demands pose significant challenges for deployment, particularly in resource-constrained environments. This project focuses on reproducing and extending the work presented in "E.T.: Re-Thinking Self-Attention for Transformer Models on GPUs" [1]. This original paper introduces a novel way of implementing self-attention and an attention-aware pruning algorithm to accelerate inference on GPUs. This report details the efforts to reproduce the alternative self-attention mechanism and the tile-based pruning process, along with preliminary experimental results and analysis. The study confirms the potential of the proposed techniques to improve the efficiency of Transformer models and recommends further research to optimize these models for various hardware platforms.**

1. Introduction

The advent of Transformer models [2] has revolutionized the field of natural language processing (NLP), achieving state-of-the-art results in a wide range of tasks, including machine translation, text summarization, and question answering. Transformers rely solely on a mechanism called self-attention to capture relationships between words in a sequence. This approach enables parallel computation and improved performance, particularly on long sequences. Self-attention is increasingly being used in other domains as well, such as for image generation tasks with diffusion models, in graph neural networks, and elsewhere.

Despite their success, Transformer models suffer from two major drawbacks: their gigantic model size and prolonged turnaround time. These challenges stem from the quadratic computational complexity of the self-attention mechanism with respect to the sequence length and the large number of parameters in these models. For instance, models like BERT [11] and GPT-3 [3] have hundreds of millions or even billions of parameters, making them difficult to deploy on resource-constrained devices.

To address these challenges, Chen et al. [1] introduced "E.T." in their paper, E.T.: Re-Thinking Self-Attention for Transformer Models on GPUs, which proposes a novel self-attention implementation mechanism and an attention-aware pruning algorithm designed to accelerate inference on GPUs. This project aims to reproduce a subset of the experiments presented in the original paper, specifically focusing on the alternative self-attention implementation and the tile-based pruning process.

The input to Transformers is an array of tokens. These tokens are mapped to representative embeddings with positional encodings. This process results in a matrix, denoted as $\mathbf{X} \in R^{L\times d}$, where L is the sequence length and d is the embedding dimension. For single-head attention, the query, key, and value matrices are computed as:

$$\mathbf{Q} = \mathbf{X}\cdot\mathbf{W}_Q^T, \quad \mathbf{K} = \mathbf{X}\cdot\mathbf{W}_K^T, \quad \mathbf{V} = \mathbf{X}\cdot\mathbf{W}_V^T$$

with learnable weights $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in R^{d_k\times d}$. The attention output for a head is calculated as

$$\mathbf{Z}_1 = \text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\cdot\mathbf{K}^T}{\sqrt{d_k}}\odot\mathbf{M}\right)\cdot\mathbf{V}$$

where $\odot$ represents element wise matrix multiplication and $\mathbf{M}$ is a mask to prevent attending to specific tokens (e.g., future tokens in causal attention). We have $Z_1 \in R^{L\times d_k}$. For multi-head attention, outputs from n heads, $\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_n$, are concatenated along the embedding dimensions to obtain $Z \in R^{L\times(n\cdot d_k)}$:

$$\mathbf{Z} = \text{Concat}(\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_n)$$

Finally, the output of the multi-head attention is computed as:

$$\text{Output} = \mathbf{Z}\cdot\mathbf{W}_O^T$$

where $\mathbf{W}_O \in R^{d\times(n\cdot d_k)}$ is a learnable projection matrix.

Usually, when attention is implemented in code, it is implemented in a modular way following the principles of software engineering. This design leads to two performance issues: (i) during consecutive matrix multiplication, one has to transfer the output of one operator to another in the GPU global memory, and (ii) this switching of operators introduces on- and off-chip data movement. Problem (i) has been solved via kernel fusion by NVIDIA's TensorRT. However, the second issue remains since TensorRT cannot change operator implementation. This paper solves this problem by implementing attention in a single operator. Here, an operator means a CUDA kernel.

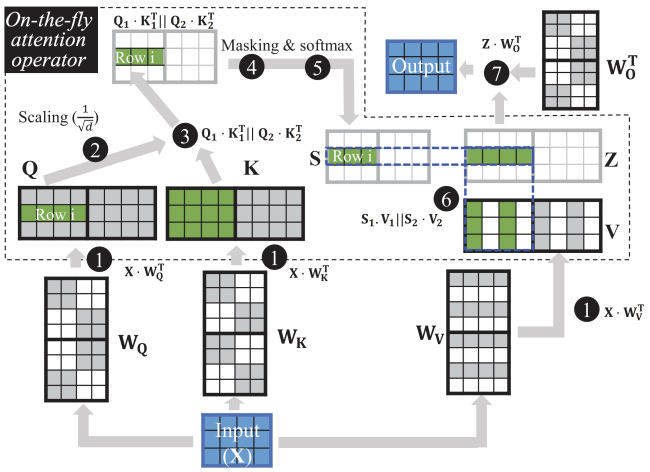


Figure 1. On the fly attention mechanism introduced by the paper [1]. The paper proposes steps (2), (3), (4), (5) and (6) to be done sequentially without swapping the shared memory. The figure shows a sequence of three tokens, four features per token and two heads each of which represented by a thicker border box.

2. Methods

This project focused on reproducing two key aspects of the E.T. paper:

2.1. Alternative Self-Attention Implementation

The original E.T. paper [1] proposes an optimized on-the-fly self-attention operator that performs five operations (matrix multiplications, scaling, masking, softmax, and another matrix multiplication) in a single GPU kernel. The overall optimized process is shown in figure 1. While computing self attention, after $\mathbf{Q}, \mathbf{K}$ and $\mathbf{V}$ matrices are computed, in step (2) the scaling by $\frac{1}{\sqrt{d_k}}$ is done on Q before further computation unlike in a normal implementation where this is done after matrix multiplication. This is done to avoid floating point overflow and it enables to fully use FP16 (half precision) for all operations. In the third step, matrix multiplication is performed between a subset rows of Q and K matrix. Here a set of rows from a head of Q are stored in shared memory, and a head of K is loaded from global memory to calculate intermediate result $\frac{Q_1 \cdot K_1^T}{\sqrt{d_k}}$ which is stored in shared memory. Then within the shared memory, masking and softmax calculations are done from that subset of rows. Finally corresponding head of V is loaded from global memory to compute a set of rows of Z. This approach reduces data movement between global memory and shared memory/registers, leading to significant performance gains.

The above approach has a limitation: as the sequence length, i.e., the number of columns in one head of $\mathbf{K}$, increases, it may not be possible to load the entire matrix into shared memory to perform matrix multiplication in steps 3 and 6. In such cases, E.T. processes the computation in parts by first loading the first column of $\mathbf{Q}$ and the first row of $\mathbf{K}^T$ to compute partial results for the entire $\mathbf{Q_1 K_1^T}$. Subsequently, it processes the follow-up columns of $\mathbf{Q_1}$ and rows of $\mathbf{K_1^T}$. This design ensures that $\mathbf{Q_1}$ and $\mathbf{K_1}$ are loaded only once. However, it requires scheduling the entirety of the GPU to perform this computation, writing the results to global memory, performing global synchronization, and then loading one row of $\mathbf{Q_1 K_1^T}$ into shared memory for masking, softmax, and eventual multiplication with $\mathbf{V}$. Matrix multiplication is performed using outer product. This results in drop in performance, a limitation observed in experiments as the sequence length increases.

2.2. Tile-Based Pruning

The proposed attention-aware tile-based pruning methodology compresses Transformer models by strategically removing tiles (sub-matrices) from the weight matrices of the self-attention mechanism. This process involves four stages: partitioning weight matrices into non-overlapping tiles, calculating tile importance using attention-aware metrics, pruning low-importance tiles based on a threshold, and retraining the pruned model to restore accuracy. The weight matrices $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ are divided into tiles of a predefined size (e.g., 16x16), aligning with hardware architecture for optimal performance. Tile importance is quantified using an L2 norm-based metric, calculated as the square root of the sum of squares of all elements within the tile, providing a measure of the tile's overall magnitude. This metric is further refined with re-weighted group lasso regularization during training, which adds a penalty term to the loss function that is proportional to the L2 norm of each tile, effectively encouraging less critical tiles to have smaller magnitudes and thus be more likely candidates for pruning. This regularization helps to identify and eliminate tiles that contribute less to the model's overall performance, improving efficiency without significant loss of accuracy. Pruning is achieved by setting low-importance tiles to zero, targeting a specific sparsity level while controlling the trade-off between compression and accuracy.

After pruning, a retraining phase compensates for accuracy loss by fine-tuning the remaining non-zero weights while keeping pruned tiles fixed at zero. This process ensures the model adjusts effectively to the reduced structure. By using structured tile pruning, the method maintains hardware-friendly regularity, reducing computational and memory demands while preserving accuracy. These advantages make attention-aware tile-based pruning particularly suited for deploying Transformer models on resource-constrained devices, achieving significant compression without sacrificing performance.

3. Experiments

Evaluation platforms. Our Training and Evaluation are performed on Python 3.6.13 and CUDA 11.2 on V100S GPU and Intel(R) Xeon(R) Gold 6238 @ 2.20GHz CPU on Palmetto Cluster. The model is trained with PyTorch 1.10.2,

and the source code of our implementation is compiled with NVIDIA nvcc 11.2 and GCC 8.5.0.

Making code work on Palmetto: The original paper utilized PyTorch 1.4.0 and GCC 7.5.0, but since this PyTorch version is no longer available through Conda channels or Python pip, we migrated the code to PyTorch 1.10.2. While the core PyTorch code remained unchanged, other dependencies, such as the Transformers package, required updates to newer versions that introduced breaking changes. This necessitated extensive code modifications. Additionally, configuring the experimental platform with a different CUDA version, unavailable through standard modules, was time-consuming to set up.

3.1. Self-Attention Performance Evaluation

For this experiment, a Transformer model with 12 encoder layers was trained on the WikiText-2 [3] dataset. The model was implemented and trained using PyTorch. Subsequently, inference evaluations were conducted using the author's CUDA/C++ single-kernel implementation of the attention mechanism. The primary objective of this experiment was to assess the performance of the alternative self-attention implementation kernel and compare the results with the claims made in the original paper. Using NVIDIA's nvprof profiler, following metrices were collected on the implemented kernel:

Memory Throughput: Analyzing the global memory read and write throughput to assess the efficiency of data movement.

Memory Transactions: Analyzing the global memory read transactions and write transactions. Transactions are a measure of latency and overhead associated with memory access. And a lower value is more preferred.

SM Occupancy: It measures the utilization of streaming multiprocessors (SMs) on the GPU. It gives the percentage of active Warps at any given time.

Instructions Per Cycle (IPC): IPC measures the average number of instructions that are executed by the kernel in each clock cycle.

These metrics were collected for sequence lengths of 64, 128, 292, 256, 320 and 364 for batch size = 1 to understand the performance characteristics of the alternative implementation similar to the original paper.

3.2. Pruning Impact on Performance and Accuracy

This experiment aimed to assess the impact of tile-based pruning on the model's performance and accuracy. The experiments were conducted with a pre-trained BERT_base model. The operation was text classification. We first fine-tuned the model using Microsoft Research Paraphrase Corpus (MRPC) dataset from General Language Understanding Evaluation (GLUE) benchmarks. The following steps were performed:

Model fine-tuning: In this step, the pre-trained BERT_base model was subjected to fine-tuning process using the MRPC dataset. This involved adjusting model's parameters through training on the labeled examples in the MRPC dataset, thereby optimizing its performance for the paraphrase identification task. The model was fine-tuned for 4 epochs with batch size of 32.

Re-weighted training: In this step, certain percentage of tiles (30% to be exact) within the model's weight matrices were encouraged to converge towards lower L2 norms. Here, training process was modified to include the re-weighting mechanism. The objective function during training was altered to incentivize these selected tiles to minimize their L2 norm during the parameter updates. Here, as well, model was trained for 4 epochs with batch size of 32.

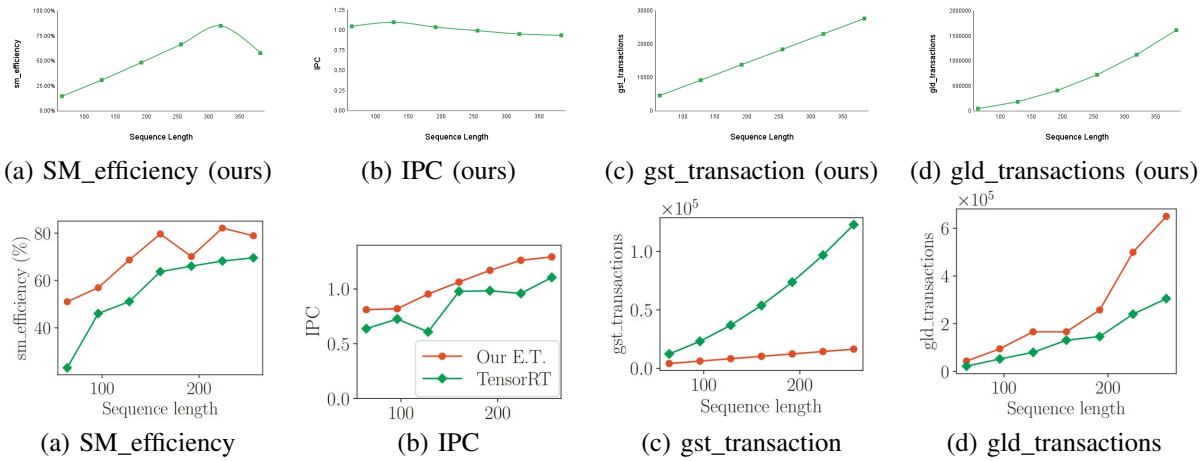
Prune and re-weighted training: In this step, based on the results of the re-weighted training in the previous step, the tiles exhibiting the lowest L2 norms were pruned from the model. After pruning operation, the reduced model, now with a smaller number of parameters, was subjected to another round of re-weighted training. It solely focused on the unpruned weights, allowing the model to adapt and compensate for the removal of the pruned weights. Here, as well, the model was fine-tuned for 4 epochs with batch size equal to 32. The pruning ratio was kept at 0.3 i.e. 30% of weights were pruned.

F1 score, an approximate harmonic mean of precision and recall, provides a balanced measure of the model's classification accuracy. F1 score was calculated along with the overall accuracy which represents the percentage of correctly classified instances in the MRPC dataset.

4. Results

The results obtained after profiling the optimized kernel are presented in Figure 2 and Figure 3. Figure 2 shows the Stream Multiprocessor(SM) efficiency, Instructions Per Cycle (IPC), Global memory store (GST) transactions and Global memory load (GLD) transaction results we obtained vs the results obtained by authors. The results suggest that the optimized performance is considerably higher than the baseline. We can observe this for the SM efficiency. We observe better SM efficiency (higher) and gst_transactions (lower) compared to their result. But when sequence length is increased beyond 320, the performance drops as observed in the graph. This is because step 3 and 6 cannot be computed as shown in figure 1. The kernel resorts to matrix multiplication via outer product as discussed in the methodology, hence resulting in drop in performance. The IPC result is comparable to the author's results. The number of gld_transactions is higher in our experimental results than in the author's.

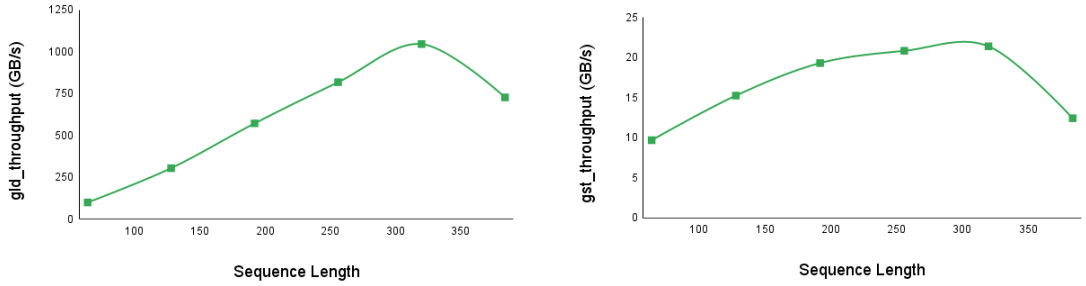
Similarly, results for the memory throughput are shown in Figure 3. The paper suggests that they were able to achieve 311 GB/s gld throughput for attention computation for sequence length of 128. We observe approximately the same throughput for that sequence length in the graph. The throughput increases gradually with increase in sequence length and drops as it goes past a certain threshold as the method resorts to using outer product for matrix multiplication on step 3 and 6.



(a) SM_efficiency (ours) (b) IPC (ours) (c) gst_transaction (ours) (d) gld_transactions (ours)

(a) SM_efficiency (b) IPC (c) gst_transaction (d) gld_transactions

Figure 2. Our result (first row) vs their results (second row) against TensorRT implementation from the paper from profiling of the optimized attention kernel. Our experiments were conducted using sequence length of 64, 128, 192, 256, 320 and 384.



(a) gld_throughput (b) gst_throughput

Figure 3. The results of computing global memory read throughput (gld_throughput) and global memory write throughput (gst_throughput) for the kernel using sequence length of 64, 128, 192, 256, 320 and 384.

Step	Eval f1	Eval Acc & f1
Before finetuning on MRPC	0.793	0.725
After finetuning on MRPC	0.986	0.9846
Re-weighted training	0.906	0.885
Prune and re-weighted training	0.7807	0.739

Figure 4. The evaluation results of the $BERT_{BASE}$ model on the MRPC dataset from the GLUE benchmark are presented for the pretrained model, fine-tuned model, re-weighted model, and pruned model.

The results of the pruning experiment we conducted are shown in Figure 4. It displays the F1 and accuracy scores for the $BERT_{BASE}$ model on the MRPC dataset from the GLUE benchmark. We can observe that when the model is pruned the F1 score decreases from 0.986 to 0.780. This is when pruning ratio is 0.3. This is a considerable loss. This result is after 4 epochs of re-training. We need to perform more experiments to draw a reliable conclusion.

5. Issues faced and Limitations of this report

This project faced significant challenges in reproducing the experiments presented in the E.T. paper [1]. These challenges primarily stemmed from the incomplete and non-functional nature of the codebase provided by the original authors. The publicly available code repository lacked crucial components of their implementation, and the existing code was not directly executable. Consequently, a considerable amount of time was dedicated to setting up the appropriate environment and modifying the codebase to achieve a working state. Due to these initial hurdles, the experiments conducted as part of this project are not as comprehensive as originally intended. Therefore, it is difficult to draw definitive conclusions that directly parallel the findings of the E.T. paper.

Furthermore, most experiments in this project were performed using a custom transformer model provided within the author's codebase. This differs from the original paper, which utilizes the BERT_base model for its primary experiments. The use of different models introduces a critical discrepancy, making direct comparisons between the results of this project and those of the E.T. paper unreliable. This project should be viewed as an exploratory study that at-

tempted to implement the core ideas presented in the E.T. paper, rather than a strict, verifiable reproduction of the original work.

6. Limitations of original paper and Further Research

While the E.T. paper presents a novel approach to self-attention optimization, it exhibits certain limitations that warrant further investigation. As noted in the results section of the E.T. paper and corroborated by our findings, their custom implementation does not yield significant efficiency gains when processing long sequences. This performance degradation, attributed to limitations in shared memory usage, represents a major bottleneck that needs to be addressed in future work. Research should focus on developing strategies to mitigate this issue, potentially through techniques like more advanced tiling or hybrid approaches combining fused-kernel operations with optimized outer product computations for longer sequences. Although attention-aware pruning is a notable strength of the paper, the exploration of pruning techniques is primarily confined to weight pruning. Future research could investigate other pruning methods, including attention head pruning and structural pruning, which involves removing entire layers or filters. This broader exploration could potentially lead to further improvements in model efficiency and compression. The current implementation relies on FP16 precision for all operations. Investigating mixed-precision approaches, incorporating INT8 and FP16, could potentially enhance efficiency without substantial performance degradation. Moreover, the implementation is tailored specifically for Nvidia GPUs. Extending the research to other hardware platforms, such as Apple M-series chips and AMD GPUs, would broaden the applicability of the proposed techniques. The current attention implementation is limited to inference. Extending the optimized self-attention mechanism to support efficient training would be a valuable contribution. The original paper, as well as this reproduction attempt, primarily evaluates performance with a batch size of 1. Future research should explore the performance implications of using larger batch sizes, which are more representative of real-world deployment scenarios.

7. Conclusion

Replicating a subset of experiments from the E.T. paper [1], this project explored an alternative self-attention mechanism and tile-based pruning for Transformer models. Our initial results affirm the potential of the on-the-fly self-attention operator to significantly improve GPU efficiency through reduced data movement and increased parallelism. We also observed a trade-off between model size reduction via tile-based pruning and the consequent accuracy drop, indicating a need for further investigation. Additionally, we presented the issues faced in replicating this work, the limitations of this implementation with longer sequence lengths, and potential further research directions.

References

[1] S. Chen, S. Huang, S. Pandey, B. Li, G. R. Gao, L. Zheng, C. Ding, and H. Liu, "E.t.: re-thinking self-attention for transformer models on gpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476138

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[3] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016.