Accelerating Compact Convolutional Transformers with FlashAttention and Triton Kernels

William Yang, Zijie Cai
University of Maryland

Abstract

With the rapid development of natural language processing, the Transformer architecture has laid a strong foundation for not only large language models but also computer vision models. However, regular Transformers often require significant amounts of training data and parameters in order to perform well. This is not very suitable for small sets of data or domain-specific applications. For instances where data availability and compute resources are limited, a simpler Convolutional Neural Network (CNN) is usually preferred for vision tasks over Vision Transformer (VIT). This leads to the development of Compact Convolutional Transformers (CCT), which replace the traditional patch embedding block with a convolutional embedding and utilize sequence pooling instead of slicing to eliminate the need for positional embeddings and class tokens, allowing for better inductive bias and accuracy with more flexible input parameters over other variations. Currently, we are drawing our inspiration from the following repository from SHI-Labs [3]. Still, these transformers incur significant memory and communication overhead. Our approach to further improve the efficiency and memory bottlenecks involves the following steps: 1. Benchmarking initial performance metrics 2. Identifying bottlenecks 3. Implement flash attention 4. Implement Triton kernel optimizations

1 Introduction

Convolutional neural networks (CNNs) have dominated the landscape of computer vision with remarkable performance. CNNs are effective for visual tasks because they manage spatial translations and incorporate strong biases inductively. They also utilize sparse interaction, weight-sharing, and equivariant representations of the model. Specifically, convolution and pooling layers provide translational equivariance and invariance, respectively, allowing these models to efficiently capture natural image statistics and achieve better sampling efficiency.

Conversely, natural language processing (NLP) has been transformed by the introduction of the transformer architecture [5]. Crafted with NLP in mind, the transformer architecture has pervaded even into computer vision with the advent of vision transformer (ViT) [2]. Although ViT showed great success, transformers still lack some of the inductive biases inherent to CNNs, such as translation equivariance and locality, and therefore do not generalize well when trained on insufficient amounts of data. The modern shift to the data-hungry paradigm makes training transformers from scratch seem intractable for many types of pressing problems, where there is significantly less data available.

The above concerns motivated the efforts for efficient models that can be effective in less data-intensive domains and allow for training on datasets that are orders of magnitude smaller than those conventionally seen in computer vision and NLP. Both Transformers and CNNs have highly desirable qualities for statistical inference and prediction, but each comes with its own costs. CCTs can both attend to important features within images while also being spatially invariant, where there are sparse interactions and weight sharing. They offer a hybrid design with convolutional embeddings, sequence pooling, and small data generalization. The backbone of the model is still transformer-based. The

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

encoder consists of transformer blocks, each including an MHSA layer and an MLP block. The encoder also applies Layer Normalization, Gaussian Error Linear Unit (GeLU) activation, and dropout. Positional embeddings can be learnable or sinusoidal, both of which are effective. Sequence pooling (Seqpool) pools the entire sequence of tokens produced by the transformer encoder, replacing the conventional class token. Figure 1 shows the CCT architecture.

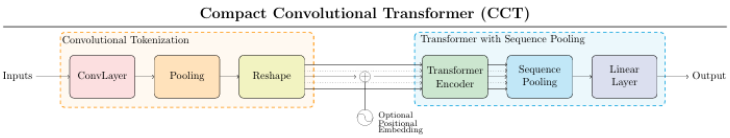

Figure 1: CCT Architecture.

Even though CCT models have shown great promise in training with small datasets such as CIFAR-10 and CIFAR-100, these models still suffer from classical bottlenecks such as multi-head attention and memory-intensive MLP layers. Our work aims to increase the efficiency of existing CCT models by introducing Flash-Attention [1] and fused, optimized Triton kernels.

Flash-Attention is a memory-efficient mechanism that avoids explicit instantiation of attention matrices. Instead, it uses a tiling strategy that fuses the softmax computation with the attention value accumulation to reduce the memory footprint from quadratic to linear in terms of sequence lengths. Flash attention also accelerates computational efficiency by loading data into SRAM, a quicker form of memory in GPU architecture. By integrating the Flash-Attention module into the transformer block of the CCT encoder, we aim to reduce both memory usage and time for training without sacrificing classification accuracy.

In addition, the MLP layers in the transformer are also a bottleneck with a heavy load of computation and memory overhead. These two layers typically consist of large matrix multiplications along with activation functions. This can slow down the training due to the separate function calls and intermediate memory access. To address this, we implement a Triton-based fused MLP pipeline with two kernels for each layer, which combines the operations of linear projection, bias accumulation, and GeLU activation into a single GPU kernel. This single fused kernel reduces unnecessary memory traffic and improve overall throughput.

By combining the two strategies of Flash-Attention and Triton-based fused MLP kernels, it provides us a faster and more efficient CCT variant that is compatible with training and inference with even larger sequence lengths and batch sizes.

2 Related Work

2.1 Transformers in Vision

Transformers were originally introduced for sequence modeling in natural language processing [5] and have shown great promise. Recently, the architectures have been adapted for vision tasks as well, with enormous success (e.g., ViT). Vision Transformers (ViT) [2] breaks down images into patches as tokens for the transformer encoder. While ViT achieves strong performance on large datasets and a diverse set of downstream vision tasks, it is not efficient on small datasets, as the transformer blocks are data-intensive. Several works have attempted to inject convolutional priors into the transformer models, such as CvT [7] and CCT [3]. CCT, in particular, is a hybrid approach combining CvT and ViT by replacing traditional patch embeddings with convolutional tokenizers and the class token with sequence pooling. These strategies make the model more robust when training with limited data, which is critical for domain-specific applications with constrained resources.

2.2 Efficient Attention Mechanisms

Traditional MHSA scales quadratically with input sequence length in both memory and computation. This is a significant bottleneck for long sequences or high-resolution images for vision tasks. To address this, several efficient attention mechanisms have been proposed, such as Linformer [6], which is an approximation attention method. More recent advances include Flash-Attention [1],

which reformulates the attention computation as a single fused kernel using tiling blocks and is much more memory efficient without explicit notation of memory sharing for all attention matrices. Flash-Attention achieves linear memory complexity while maintaining numerical stability, which makes it a popular choice to plug into a transformer block for many models and provides robustness in training speed and memory usage.

2.3 Triton Kernel Optimization

Triton [4] is a compiler language designed for writing custom GPU kernels like CUDA kernels, but with a Python-like syntax which offers both high performance and user-friendly flexibility. It enables developers to write fused kernels that combine multiple PyTorch operations into a single pass to eliminate memory overhead and improve overall throughput. There have been many tutorials on using Triton to accelerate matrix multiplication and activation functions. In our work, we adopt Triton to fuse the two-layer feed-forward MLP layers into a single-stage pipeline, specifically for linear layers and GeLU activations together. This helps us reduce kernel launch overhead and redundant memory access, leading to better GPU utilization.

3 Methodology

3.1 Compact Convolutional Transformer Architecture

Our work is based on the Compact Convolutional Transformer (CCT) proposed by Hassani et al. [3]. The model architecture consists of three key components: a convolutional tokenizer, a transformer encoder, and a sequence pooling layer. The convolutional tokenizer replaces the standard patch embedding from ViT with multiple convolutional layers, which introduces low relational inductive bias and invariance to spatial translations. The transformer encoder consists of multi-head self-attention (MHSA) and a two-layer feed-forward MLP. These layers include a chain of operations like LayerNorm, dropout, and GeLU activation. Sequence pooling pools over the tokens from the transformer encoder, which allows the model to better utilize information across spatially sparse data.

To better understand the performance of the baseline model variant and help us identify memory and attention bottlenecks, we profiled the training loop using `torch.profiler`. We recorded the CPU and CUDA kernel timings along with detailed memory usage by each function call for the first 20 iterations of model training on the Flowers-102 dataset. Profiling for baseline shows that the majority of CUDA time is concentrated in the convolutional kernel operation, and the MLP and attention blocks also showed significant usage of memory and CUDA time in both the forward and train_partial operations. We later apply the same profiling strategies to our fused model variants, showcasing specific improvements of our implementations.

3.2 FlashAttention Integration

In the baseline CCT model, the MHSA is implemented with PyTorch's native attention function, which computes each step of the attention value matrix by explicitly storing all intermediate values for dot-product, softmax, etc. This process leads to a quadratic memory footprint in terms of sequence length, and the operations are sequentially separate function calls, which can slow down the performance due to limited memory bandwidth.

To improve this, we replace the attention value computation in each transformer block with the FlashAttention [1] module, which provides a memory-efficient CUDA kernel that packs all operations of softmax and value aggregation together into a single function call. We install the module from the official Flash-Attention Python Library. This eliminates all intermediate attention matrix storing and reduces the memory complexity from $O(n^2)$ to $O(n)$ with respect to sequence length while ensuring numerical stability. Our implementation involves adapting the transformer block to reshape the query, key, and value tensors to work with the fused Flash-Attention's kernel's input function definition. In a single fused pass, these tensors are passed through and output the final attention value. Compared to the baseline, this integration significantly reduces memory usage and latency during both training and inference, especially for long sequences or larger batch sizes.

3.3 Triton Fused MLP Kernel Design

In the baseline CCT implementation, the MLP block is executed using four separate layers in PyTorch:

```
self.linear2(self.dropout1(self.activation(self.linear1(src))))
```

Each operation corresponds to a separate function call, introducing unnecessary memory movement between global memory and registers, which results in higher latency and also more memory usage due to intermediate synchronization.

To address this, we wrote a custom fused MLP module using Triton [4]. Our `FusedMLP` module uses two Triton kernels: the first kernel (`linear_bias_gelu_kernel`) performs a fused linear transformation, bias addition, and GeLU activation; the second kernel (`linear_bias_kernel`) performs the final linear projection and bias addition.

The input tensor of shape `(B, L, H)` is reshaped into `(B·L, H)` for computation, where `B` is the batch size, `L` is the sequence length (number of tokens), and `H` is the hidden dimension. For our fused MLP module, stage 1 is the first kernel (Linear+bias+GeLU) that performs tiled matrix multiplication for the input tensor and weight tensor, then adds the bias and applies the GeLU activation, which we used its mathematical formula with tanh approximation. Dropout was applied using the original PyTorch dropout function. For stage 2, the previous result is projected back with the second kernel to the original hidden size and accumulates bias. Both kernels are parameterized by a tunable block size configuration, like with Triton's Autotune function for optimized performance.

Our approach significantly reduces memory traffic by avoiding redundant reads and writes, and reduces kernel launch overhead by using a fused kernel. Compared to the baseline, the fused MLP kernel results in less memory usage during training and therefore improves the model scalability.

```
// Pseudocode for linear_bias_gelu_kernel (Triton)
for pid_m in M_blocks:
  for pid_n in N_blocks:
    acc = zeros(BLOCK_M, BLOCK_N)
    for k0 in range(0, K, BLOCK_K):
      x_tile = load(X, offsets=(pid_m, k0))
      w_tile = load(W, offsets=(k0, pid_n))
      acc += dot(x_tile, w_tile)
    # Bias
    bias = load(B, pid_n)
    y = acc + bias
    # GeLU
    y = 0.5 * y * (1 + tanh(sqrt(2/pi) * (y + 0.044715 * y^3)))
    store(Y, pid_m, pid_n, y)
```

Figure 2: Pseudocode illustration of the fused Linear+Bias+GeLU Triton kernel.

4 Experiments

4.1 Setup

We evaluate the impact of our optimizations on the Flowers-102 image classification dataset. One thing to note about our decision to use the Flowers-102 dataset is that we performed preliminary experiments on CIFAR-10 with the smallest CCT model architecture. Although we noticed a 46% improvement in GPU time and several orders of magnitude improvement in memory, we also observed a 36% decrease in model performance. Our initial assumption led us to believe that the training dataset and model were too simple to be paired with such optimizers, leading to our decision to use a more complex training dataset and model. Thus, due to its higher resolution and larger number of classes, Flowers-102 provides a more meaningful benchmark than CIFAR-10 when assessing architectural improvements. We test on the following variant of Compact Convolutional Transformers (CCT):

- **CCT-14/7x2_384**: A wider and deeper configuration with 14 transformer layers, a 7×2 tokenizer, and a hidden size of 384. This setting reveals more visible gains, compared to a

lighter model such as CCT-7/3x_32, when applying Flash-Attention and Triton-based fused MLP due to longer sequences and higher token count.

All models are trained for 100 epochs using the AdamW optimizer, a cosine annealing learning rate scheduler, and a batch size of 8. Unless stated otherwise, all training hyperparameters are kept identical across variants for fair comparison. Training and profiling are tested on a 1/7th partition of an NVIDIA A100-SXM4-40GB GPU. We use PyTorch with native AMP enabled. All custom Triton kernels and Flash-Attention modules are fully compatible with this setup.

4.2 Evaluation Strategy

We evaluate each model along three categories:

- **Accuracy**: Measured as top-1 test set accuracy after full convergence ($\tilde{9}0$ epochs).
- **Latency**: Measured as average CUDA execution time for the forward pass and partial training steps. We record 20 iterations using `torch.profiler`.
- **Memory usage**: Peak GPU memory usage tracked via `torch.cuda.max_memory_allocated()` and verified through profiler traces.

Each model is profiled for 20 training iterations. This provides us with a detailed metric of CUDA time and memory allocation for each function. We use this information to quantify the performance impact of our integration of Flash-Attention and fused MLPs.

5 Results

5.1 Accuracy vs Baseline

Figure 3 shows the top-1 test accuracy of all four model variants. The baseline CCT achieves the highest accuracy on Flowers-102. Replacing the MLP with a fused Triton implementation introduces a small drop. Flash-Attention introduces a much larger accuracy gap. The combined model achieves the lowest accuracy at 68%, which is over 20% points lower than the baseline.

We suspect that the accuracy drop may be due to: (1) the tanh-based approximation of GeLU used in our Triton kernel, which differs from PyTorch's native implementation; and (2) precision mismatches introduced by AMP or fused kernel behavior. While not ideal for pure accuracy, these trade-offs provide significant improvements in speed and memory efficiency.

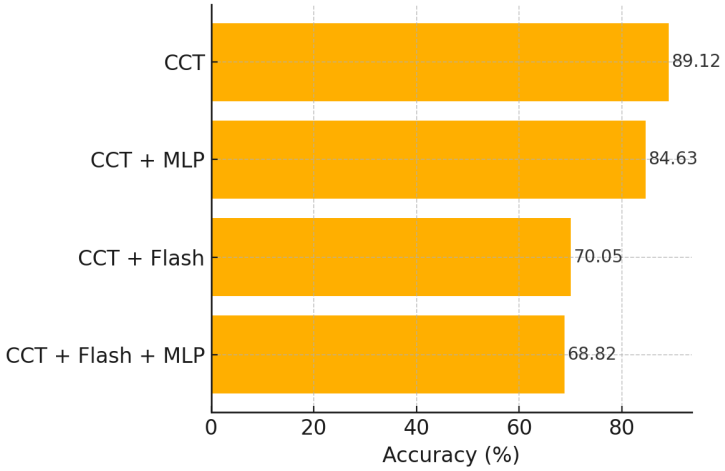

Figure 3: Top-1 accuracy of model variants on Flowers-102.

5.2 Latency and Memory Benchmarks

Table 1 shows average function latency across 20 iterations. Triton-MLP alone shows minimal improvement. The introduction of Flash-Attention, however, significantly reduces both forward and training time. The combined model achieves the lowest latency across both metrics.

Table 1: Average latency (in milliseconds) over 20 iterations.

Model Variant	Forward	Train_Partial
CCT (Baseline)	253.3	5325.0
CCT-MLP (Triton)	252.5	5308.0
CCT-Flash (FA Only)	218.1	4568.0
CCT-Flash + MLP	**217.1**	**4481.0**

Table 2 shows peak memory usage. The baseline model uses the most memory due to intermediate storage and kernel overhead. Triton-MLP alone cuts memory usage moderately. Flash-Attention provides the largest reduction. When combined, the fused model yields a 4.2× drop in forward memory usage and nearly 2× reduction during the training step.

Table 2: Peak GPU memory usage (in GB) during execution.

Model Variant	Forward	Train_Partial
CCT (Baseline)	47.20	0.301
CCT-MLP (Triton)	38.42	**0.159**
CCT-Flash (FA Only)	17.38	0.172
CCT-Flash + MLP	**11.26**	**0.159**

6 Discussion

Our results show a clear trade-off: fusing operations improves runtime and memory usage but also hurts the accuracy. In terms of accuracy, there is a 29% decrease from the baseline results compared to CCT + Flash + MLP. Our suspicion for the decreased accuracy are twofold. One source of the issue could be the mixed precision parameter. Mixed precision aims to increase training efficiency by decreasing 32-bit floating-point calculations to 16-bit floating-point calculations, while incurring only a slight drop in accuracy. The baseline results do not use mixed precision; however, either AMP or the fused kernels introduced in either flash attention or the Triton kernels could implicitly utilize this technique. Secondly, the tanh-based approximation of GeLU used in our Triton kernel differs from the baseline PyTorch native implementation.

The fused kernels minimize overhead and memory access, but numerical differences and mixed-precision artifacts may affect convergence. Specifically, we observed an 18% decrease in training time and a 300% decrease in memory usage with CCT + Flash + MLP compared to baseline. It seems that these gains mainly come from the use of flash attention due to optimized memory storage and the removal of intermediate tensors in the backward pass. There is only a slight decrease in training time, likely due to the already optimized nature of CCT, which are tiny in model size compared to state-of-the-art transformers. Overall, further investigation is required as we need to explore performance gains with even more combinations of model size and training data. For real-time inference or constrained deployments, these trade-offs could be acceptable.

In future work, we plan to extend fusion to layer normalization, dropout, and backward passes. This would allow for a fully fused transformer block in both forward and backward training stages, further improving training time. We would also like to perform more studies to determine the source of the accuracy drop. Specifically, we would like to isolate the mixed precision parameter and the fixed GeLU implementation. Lastly, we would like to explore an even larger dataset, specifically ImageNet, which contains 14,000,000 images of 2848 x 42,000 pixel size.

7 Conclusion

In this work, we explored performance and memory optimizations for CCTs through the integration of Flash Attention and Triton-based fused MLP kernels. We aimed at enhancing the training and inference of longer sequences and high-resolution image data while preserving the efficiency and lightweight nature of CCT models. Flash attention reduced the memory footprint of multi-head self-attention by avoiding storage of intermediate tensors, while our custom Triton kernels fused the MLP layers into a single GPU pass to reduce memory traffic and kernel launch overhead. Our experimental results on the Flowers-102 dataset demonstrated gains in both latency and peak memory usage. However, we also observed a noticeable trade-off in accuracy, most likely due to the integration of flash attention. This decrease in accuracy suggests that further tuning and investigation are needed. Despite the trade-offs, this work showcases the practicality and impact of low-level GPU kernel fusion and efficient attention mechanisms in accelerating CVTs. Our findings encourage future work in combining architectural efficiency with numerical stability to unlock the full potential of transformer-based vision models in resource-constrained settings.

References

[1] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16144–16159, 2022.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021. arXiv:2010.11929.

[3] Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers. *arXiv preprint arXiv:2104.05704*, 2021.

[4] Philippe Tillet, H. T. 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*, MAPL 2019, page 10–19, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, volume 30. Curran Associates, Inc., 2017.

[6] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

[7] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 22–31, 2021.