Token Merging Perceiver

Matthew Zhou

April 30, 2024

1 Introduction

With the recent boom in popularity of large language models such as ChatGPT and LLaMA, the transformer architecture has become ubiquitous in such applications. One of the main drawbacks of the transformer architecture is that the attention mechanism, which is the backbone of the model, has a runtime complexity that is quadratic in the input sequence length. This leads to the transformer architecture not being able to scale well to larger sequence lengths. In the case of large language models such as ChatGPT, this means that the model can often "forget" information from previous conversations or information that was stated far in the past, simply because the input sequence length must be bounded because of the quadratic time complexity.

To solve this issue, the Perceiver I/O architecture[2], which has a linear complexity with respect to the input sequence length, was proposed in 2022 by DeepMind. It accomplishes this by projecting the input tokens to a latent space and performing the bulk of the computation in this latent space. Furthermore, the Perceiver I/O architecture can handle input and output shapes of arbitrary shape. However, this model has limitations in that it is only performant with a large number of latent tokens. To solve this issue and create a model that is both efficient and able to handle arbitrary input and output shapes, I utilize Token Merging[1], introduced by Bolva et al. in 2023, which uses dot product similarity to merge similar tokens in each layer of a transformer model. In particular, I apply this technique to the latent tokens of the Perceiver I/O to remove redundancy between latent tokens. This is able to maintain the accuracy of the model, and when retrained, can even surpass the accuracy of the original Perceiver model.

2 Existing Works

The Perceiver I/O model[2] tames the quadratic time complexity of the attention mechanism by mapping the input tokens to a set of latent tokens using cross attention. The number of latent tokens is typically 256/512, which is similar to the input sequence length of LLMs such as BERT, which has a maximum input sequence length of 512. Once the inputs are mapped to the set of latent tokens, self attention is performed on these latent tokens L times, where L is the number of layers. This is similar to a regular transformer model, where the input tokens are just replaced with the latent tokens. Finally, the output is mapped into the output space through a decoder that uses cross attention again. The benefit of this architecture is that the cross attention operations are linear in the input and output sequence length, and the runtime of the self-attention layers are independent of the input and output sequence length. This allows the Perceiver I/O to handle much longer input sequence lengths. For language tasks, it can avoid using tokenization all together and just feed in raw UTF-8 bytes. For vision tasks, full images can be fed in instead of patches of images. Finally, the Perceiver I/O can handle multi-task queries through the use of task-specific tokens.

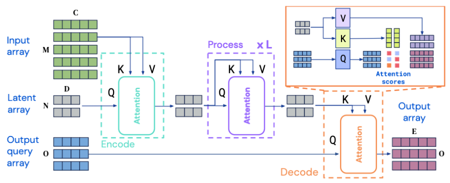


Figure 1: Perceiver IO Architecture

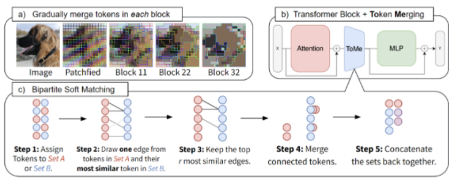

a) Gradually merge tokens in *each* block
Image
Patchfied
Block 11
Block 22
Block 32
b) Transformer Block + **Token Merging**
x
Attention
ToMe
MLP
y
c) Bipartite Soft Matching
Step 1: Assign Tokens to *Set A* or *Set B*.
Step 2: Draw **one** edge from tokens in *Set A* and their **most similar** token in *Set B*.
Step 3: Keep the top *r* most similar edges.
Step 4: Merge connected tokens.
Step 5: Concatenate the sets back together.

Figure 2: Token Merging

Token Merging[1] is a technique to merge similar tokens. This is conceptually similar to pruning, where only the most similar tokens are kept in order to increase the efficiency and generalizability of the model. However, while pruning is applied to the weights, Token Merging is applied to the input tokens of each layer. Specifically, r tokens are merged each layer, so over L layers, rL tokens in total are removed. The metric to measure similarity between is dot product similarity. In order to maintain efficiency, this merging operation needs to be fast. So, Token Merging uses Bipartite Soft Matching, which is a fast approximate matching algorithm to determine similarity. It splits the tokens into two sets, and for each token in the first set, it draws an edge to the most similar token in the second set. Then, the r most similar edges are kept, and the tokens that are connected by an edge are merged. This operation is also similar to a pooling layer applied after attention and before the next layer.

There is not any existing works on performing token merging or pruning to the Perceiver I/O. This is mainly due to the lack of popularity of the Perceiver I/O model. The limitation of the Perceiver I/O comes from the high number of latent tokens needed to achieve high performance. This makes the Perceiver I/O still computationally expensive, which defeats the purpose of the model.

3 Proposed Solution & Experiments

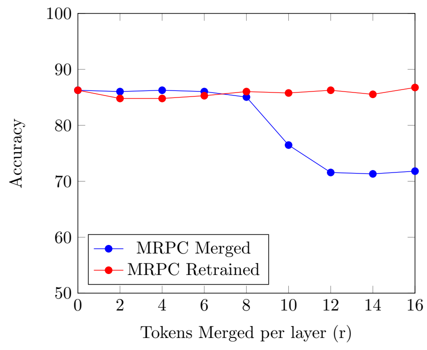
My proposed solution is to utilize Token Merging on the Perceiver I/O to merge the latent tokens. This reduces the number of latent tokens that the self-attention layers operate on. This reduces the number of FLOPs of the Perceiver, making it both able to handle longer and arbitrary input sequences, while achieving the original goal of being efficient regardless of input size.

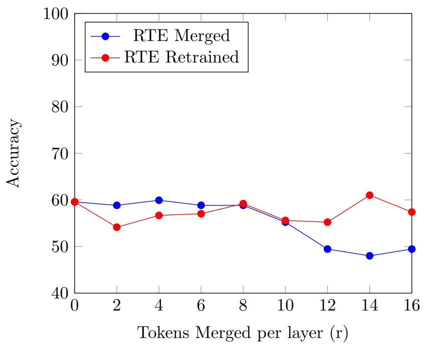
To test this solution, I finetune the Perceiver I/O model on the GLUE dataset and FashionMNIST, to demonstrate that my method applies to both language and vision tasks. After finetuning, I apply Token Merging to the finetuned models, and vary r, the number of reduced tokens. I record the accuracy, and then retrain with the Token Merging layer still present for a very small number of epochs.

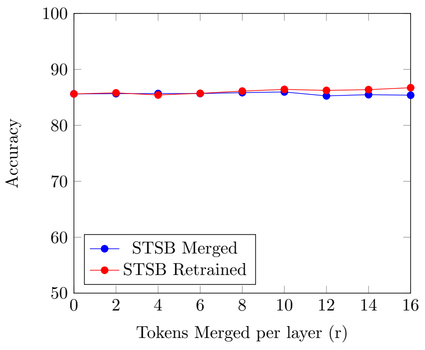
All code is available on Github.

3.1 GLUE Experiments

I run experiments on only 3 GLUE tasks due to resource constraints: MRPC, RTE, and STSB. I finetune a Perceiver I/O model pretrained on large text corpus obtained by combining English Wikipedia and C4. After running hyperparameter search on a small grid, the finetuning hyperparameters are {Epochs: 20, Learning Rate: 2e-5, Weight Decay: .01, Warmup Steps: 200, Max Sequence Length: 2048}. Retraining is done with the same hyperparameters for 5 epochs only. The batch size varies per task as follows, {MRPC: 32, RTE: 16, STSB: 32}. I vary r, the number of reduced tokens, from 2 to 16. The results are below.

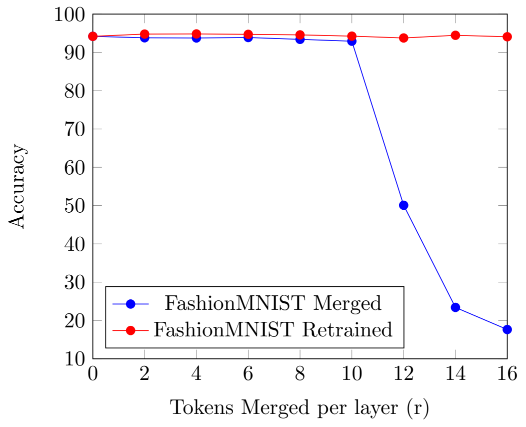






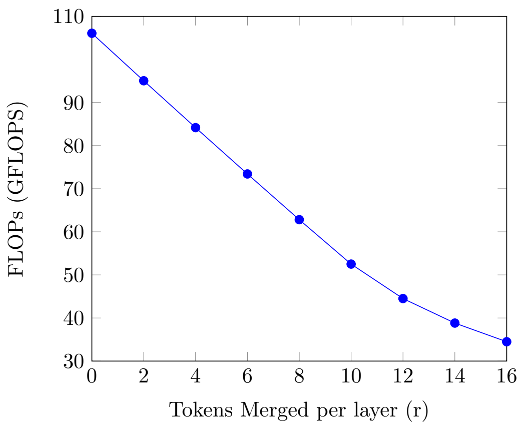
3.2 FashionMNIST Experiments

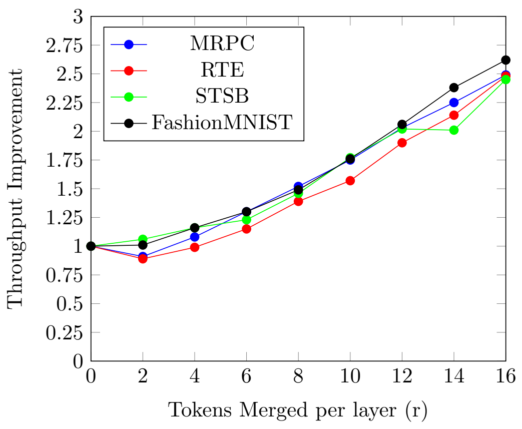
I similarly finetune a Perceiver I/O model pretrained on ImageNet on FashionMNIST. After running hypeparameter search on a small grid, the finetuning hyperparameters are {Epochs: 2, Learning Rate: 5e-5, Weight Decay: .01, Warmup Steps: 200, Max Sequence Length: 2048}. Retraining is done with the same hyperparameters for only one epoch. The results are below.



3.3 Efficiency Improvements

To calculate the efficiency improvement, I measure the FLOPs for all models, as well as the throughput speedup for each model.





4 Analysis

The results show that by simply merging the tokens after training, I only lose around 1% accuracy for up to 8 tokens merged per layer. Since there are 256 latent tokens and 26 self-attention layers in for the language tasks, this merges 208/256 tokens by the end of the model. Similarly, there are 512 latents and 48 self-attention layers for the vision tasks, which merges 384/512 of the latent tokens. This achieves a throughput speedup of around 1.5x, with the FLOPs decreasing to 62.82 GFLOPS from the original 106.08 GFLOPS. When retraining is done, I can maintain, and even improve the accuracy for all values of r. This leads to a 2.5 throughput improvement for $r = 16$, and a reduction of FLOPs to just 34 GFLOPS.

Because this method works for both language and vision tasks, I show that the Perceiver I/O's ability to handle multimodal inputs is still maintained when a large number of latent tokens are merged. This also shows that many of the latent tokens are redundant, as just a small number of tokens can encode all of the necessary information of a large input sequence. This work has many applications, as the linear time complexity allows the model to scale to very large inputs of any shape or size. By merging the tokens using Token Merging, the efficiency of the Perceiver is greatly increased while still providing similar accuracy.

References

[1] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster, 2023.

[2] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver io: A general architecture for structured inputs & outputs, 2022.