| ILLINOIS INSTITUTE OF TECHNOLOGY DEPARTMENT OF COMPUTER SCIENCE |
| --- |
| Computational Comparison of Token-Pruning Methods in Transformer Models |
| CS 577: Deep Learning |
|  |
| **Edman Alicea and Cassandra Carlson**  **Final Project Report** |
| **1 December 2023** |

**Github For Project Files**

[**https://github.com/edmangalicea/cs577FinalProject**](https://github.com/edmangalicea/cs577FinalProject)

# Introduction and Problem Description

Transformers are a very powerful model for NLP tasks because they are able to capture long-range dependencies in a given text to understand the context between words in the passage. However, transformers are also computationally expensive to train as model sizes and input sequences are becoming increasingly larger. In fact, transformer inference cost increases quadratically with input sequence length. This means that their superior performance often brings rise to delay issues that makes them difficult to deploy in latency-critical situations. Implementing transformer models in resource-constrained edge devices is an important goal for the widespread use of these models; therefore, several methods have been proposed to improve transformer performance.

For our project we decided to go with the application of a machine learning algorithm to solve a real world problem. We were drawn to focus on transformer models due to their current popularity, and the advantages they have over other models. After looking through a corpus of research text we decided to apply token pruning methods for our project. This was due to the potential upsides in inference speed and model size.

The most common techniques for increasing transformer inference speed are pruning, knowledge distillation, and quantization. Each method comes with its own tradeoffs in regards to latency and accuracy, but the overall focus of these techniques is to reduce transformer size while maintaining performance. The methods we want to focus on in this project deal specifically with the pruning of input tokens. In a transformer model, certain tokens become unimportant for inference as they pass through the different layers of the model, and removing these unnecessary tokens can substantially improve latency while preserving accuracy.

Current token pruning methods fall into two classes: attention based scoring and prediction module. Attention based scoring methods use the self-attention mechanism of transformers to rank token importance and make pruning decisions based on these rankings. These models are successful at reducing latency, however there can be large impacts on accuracy because essential tokens could receive low attention scores in the first few layers and be pruned before they are used later for inference. The second class of token pruning methods inserts another neural network prediction module at each level of the transformer to determine token importance scores more accurately. These models make great improvements to overall accuracy of the transformer, but also add a considerable amount of inference latency overhead (~30%).

Here is a brief description of some of the previous token pruning algorithms that other papers have implemented. The first of these algorithms is PoWER-BERT. PoWER-BERT utilizes attention based scoring to perform layer-wise pruning of input tokens. It first learns the optimal number of tokens to keep at each layer of the transformer and then ranks token attention values to eliminate the correct number of inputs. Learned Token Pruning, or LTP, is another attention based algorithm. LTP learns an attention score threshold at each layer and prunes all tokens below that threshold. This results in a variable input length at every transformer block and eliminates the expensive computation required to decide a fixed length at each layer before pruning. Token Pruning, or ToP, is the final attention based algorithm used for comparison. It improves upon the issue of inaccurate token ranking in early layers by distilling token rankings from the final, unpruned model and implementing them into the early layers of the model to be pruned. ToP is also given a computational constraint and controls the number of layers where tokens are to be pruned according to this metric. Another token pruning algorithm is Transkimmer which uses a prediction module. It implements a 2-layer neural network at the beginning of each transformer block to learn the importance of each token and prune them accordingly.

For this project, our team decided to use the CoFi pruning algorithm on task specific fine tuned bert models and compare its accuracy and speed up to the original model that the author’s of the CoFi paper trained. CoFi (Course- and Fine-grained Pruning) is a Task-Specific Pruning approach which combines both coarse-grained (e.g., layers) and fine-grained (e.g., heads and hidden unit modules) pruning strategies. To do so we control the pruning decision of each parameter with masks of different granularity. What sets CoFi apart is its combination of layerwise distillation with structured pruning. It learns mask variables during training, which enables precise control over sparsity levels of the model. Then we use a layerwise distillation strategy to transfer knowledge from unpruned to pruned models during training.

Task-Specific Unstructured Pruning does not use additional unlabeled data for general distillation and therefore has numerous advantages over other methods. This particular way of reducing model size simplifies the process and makes it easier to apply the pruning technique without any additional resources. CoFi is limited to Task-Specific Pruning due to the complexity of the design choices for upstream pruning, which affects the scope of possible applications and scenarios it can be utilized in.

The implementation of CoFi is described below:

We allow pruning MHA and FFN layers explicitly along with fine-grained units by introducing two additional masks zMHA and zFFN for each layer. Now the multi-head self attention and feed-forward layer become as shown in figure 4. With these layer masks, we explicitly prune an entire layer, instead of pruning all the heads in one MHA layer.CoFi differs from previous pruning approaches in that multiple mask variables jointly control the pruning decision of one single parameter. For example, a weight in an FFN layer is pruned when the entire FFN layer, or its corresponding intermediate dimension, or the hidden dimension is pruned. To learn the mask variables, we use l0 regularization modeled with hard concrete distributions

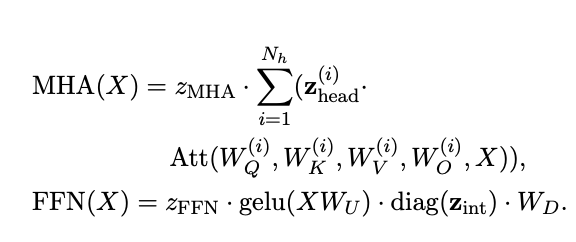


Figure 4

# Methods

For our experiments we used as reference, the CoFi source code from github. Our idea was to compare the results of CoFi pruning methods on the same task and with different models. We used a subset of the GLUE tasks particularly: MNLI, CoLA, and SST-2.

All the experiments we did we used the BERT model as a base. Specifically we used the [bert-base-uncased](https://huggingface.co/bert-base-uncased) model from hugging face, which is the 140 M parameter version and not the 340 M one. We then used a task specific tuned version of BERT to perform ToP and CoFi pruning. We attempted to use the non-task specific version of BERT for this but ran into significant slowdowns, extremely long compute times and frequent crashes. We thus decided to only use task specific versions of BERT to alleviate the extremely long compute times that were becoming a problem due to the deadline of this project, the increasingly less time we had to complete them, and the shrinking credit utilization we had on the [Google Cloud](https://cloud.google.com/?hl=en).

The datasets that we used to evaluate this project come from the GLUE (General Language Understanding Evaluation) benchmark. GLUE contains a variety of tasks that test different aspects crucial for NLP. It’s routinely used by researchers and developers in the industry to assess the performance and quality of different NLP models.

The MNLI (Multi-Genre Natural Language Inference) task is one of the components of the GLUE benchmark. MNLI specifically is a natural language inference task where the model is required to predict whether a hypothesis is entailed by, contradicts, or is neutral to a given premise. The CoLA (Corpus of Linguistic Acceptability) is another task that we used for model evaluation. In CoLA, the goal is to determine whether a sentence is linguistically acceptable in terms of syntax, grammar, and overall structure. This task specifically involves binary classification, where sentences are labeled as either grammatically acceptable or unacceptable. The final task we used was the SST-2 (Stanford Sentiment Treebank). The SST-2 task evaluates a model’s ability to correctly classify sentences as either positive or negative based on the sentiment they convey. This task is valuable for evaluating how well a model comprehends and categorizes sentiments expressed in natural language text.

**What Has Been Done So Far**

For setting up the project we have run into a couple of hurdles. Firstly the devices that we have access to in order to run model training and testing are without a CUDA device, specifically Edman has an M1 mac which not only makes model inference practically impossible but extremely time consuming since any model training/testing will happen on the CPU. We attempted to edit the repositories in order to use different backends, but all the projects that we looked at had multiple layers of checks and balances that would require a complete rewrite of the source code in order to run on any of our machines. (The MPS backend which uses the M1 gpu is not feature complete and ran into a bunch of errors that Edman attempted to fix). Our team could attempt to get access to a VM that had a CUDA device attached to it. We decided to use a cloud provided for such a service. We looked at various different companies such as Google, Amazon, Serverless, etc knowing that most of these services would charge hourly to use a GPU. We decided on Google as the cloud service provider we will go with. This was due to the fact that google provided a generous amount of free credits for new accounts at a total of $300 which was extremely substantial and should last us well beyond the entire scope of the project

We are using Google Cloud Compute which is a service offered by Google Cloud which gives us access to VMs with a wide range of configurations. The setup process was a bit challenging for us. Firstly because we have never had experienced using a cloud provided to host a VM the most we have gotten to was ssh into the server cluster on campus. Secondly, using Google Cloud and the interface they provided had its own learning curve that we had to go through. Thirdly, choosing the configuration was more difficult than we had planned.

Whenever we set up a new VM in Google Cloud, we had to install practically everything: nvidia driver, cuda, python, python packages–which had varying levels of dependencies not only on hardware but also the software. When it came to choosing a configuration for our VM we had a choice of GPU options, unbeknownst to us, the option we first chose had a host of problems. This was due to an unresolved issue between the nvidia driver and python that persisted through numerous VM restarts and factory resets. This was confirmed by us digging deep into the issue on google and finding loads of people with the same issue as us with the same configuration on Google Cloud.

We resolved this issue by setting up a new VM with the following configuration: CPU: 8 Virtual Cores, Memory: 16 GB, Disk: 128GB Persistent, OS: Debian, GPU: NVIDIA Tesla P100. After we setup the VM we proceed to setup the environment by installing python and anaconda. We will be using Conda in between projects to manage all the dependencies.

We have used up about $46 of our $400 credit that we have which should be sufficient to train the rest of our models and get the training results. Since we would like to compare models with each other, we trained and benchmarked them all on the same VM to ensure consistency in our results.

## Results

## ***CoFi***

For our first set of tests we used CoFi pruning for 3 different tasks: MNLI, CoLA, and SST-2. We used the task fine tuned version of BERT for every single task and plotted its metrics versus the steps/epochs. The BERT models we found were sourced from Hugging Face and no further information about the hyperparameters, eval loss or any other metric was listed on the website, which is a potential downside of our project. In the future we would have wanted to fine tune our own version of BERT and use that to create task specific submodels, but the computational barriers and remaining time proved substantial. All the models were pruned with a sparsity of 0.95. In our experiments, sparsity is computed as the number of pruned parameters divided by the full model size (embeddings excluded).

### *CoFi - MNLI*

The first task that we ran was [MNLI](https://openreview.net/pdf?id=rJ4km2R5t7) with a fine tuned version of BERT called: [bert-base-uncased-finetuned-mnli.](https://huggingface.co/senfu/bert-base-uncased-finetuned-mnli) We could have trained our own BERT model but that would have gone against the aims of our project, which was to understand and benchmark various pruning methods for transformer models. We wanted to confirm that the accuracy and speedup improvements that the authors claimed were substantial. So we went with a pre-trained model, however we do not have access to any of the hyperparameters that the pre-trained model used.

CoFi itself uses different parameters that are explained in detail in the paper. These are used to prune the mode and reduce its size. We list the parameters for pruning CoFi below. They are inside the run.sh file

TASK=MNLI

SUFFIX=sparsity0.95

EX\_CATE=senfu/bert-base-uncased-finetuned-mnli

PRUNING\_TYPE=structured\_heads+structured\_mlp+hidden+layer

SPARSITY=0.95

DISTILL\_LAYER\_LOSS\_ALPHA=0.9

DISTILL\_CE\_LOSS\_ALPHA=0.1

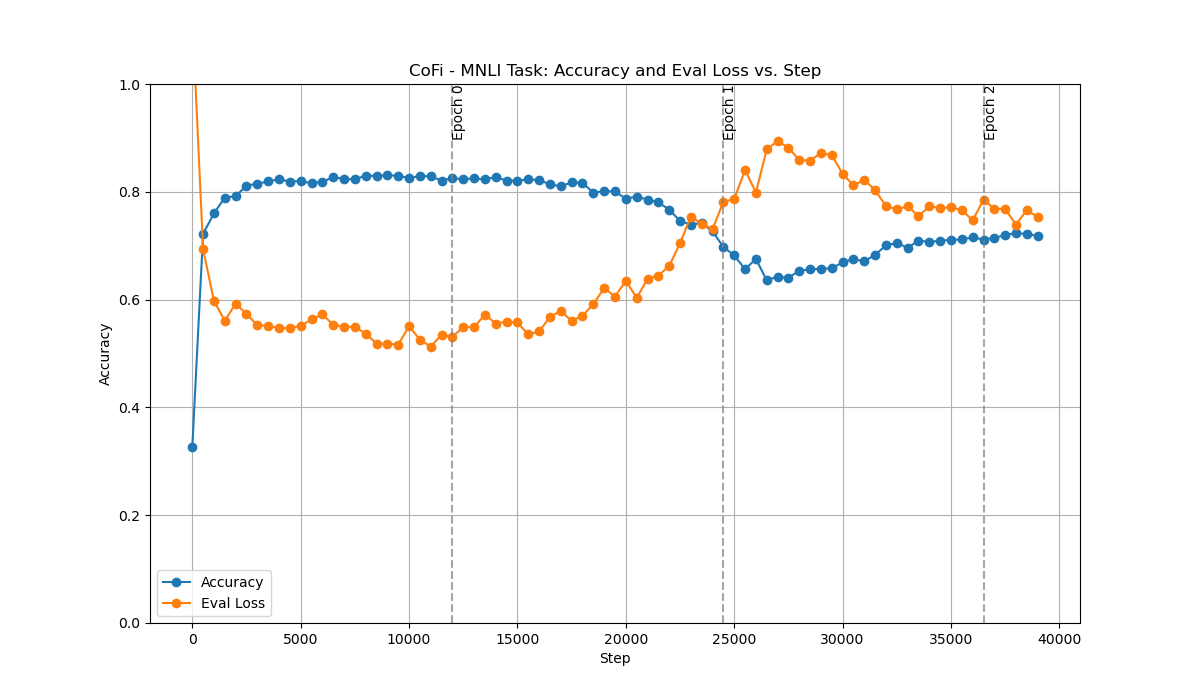
LAYER\_DISTILL\_VERSION=4

SPARSITY\_EPSILON=0.01

DISTILLATION\_PATH=senfu/bert-base-uncased-finetuned-mnli

bash scripts/run\_CoFi.sh $TASK $SUFFIX $EX\_CATE $PRUNING\_TYPE $SPARSITY $DISTILLATION\_PATH $DISTILL\_LAYER\_LOSS\_ALPHA $DISTILL\_CE\_LOSS\_ALPHA $LAYER\_DISTILL\_VERSION $SPARSITY\_EPSI

*Figure 1: CoFi Pruning Parameters*

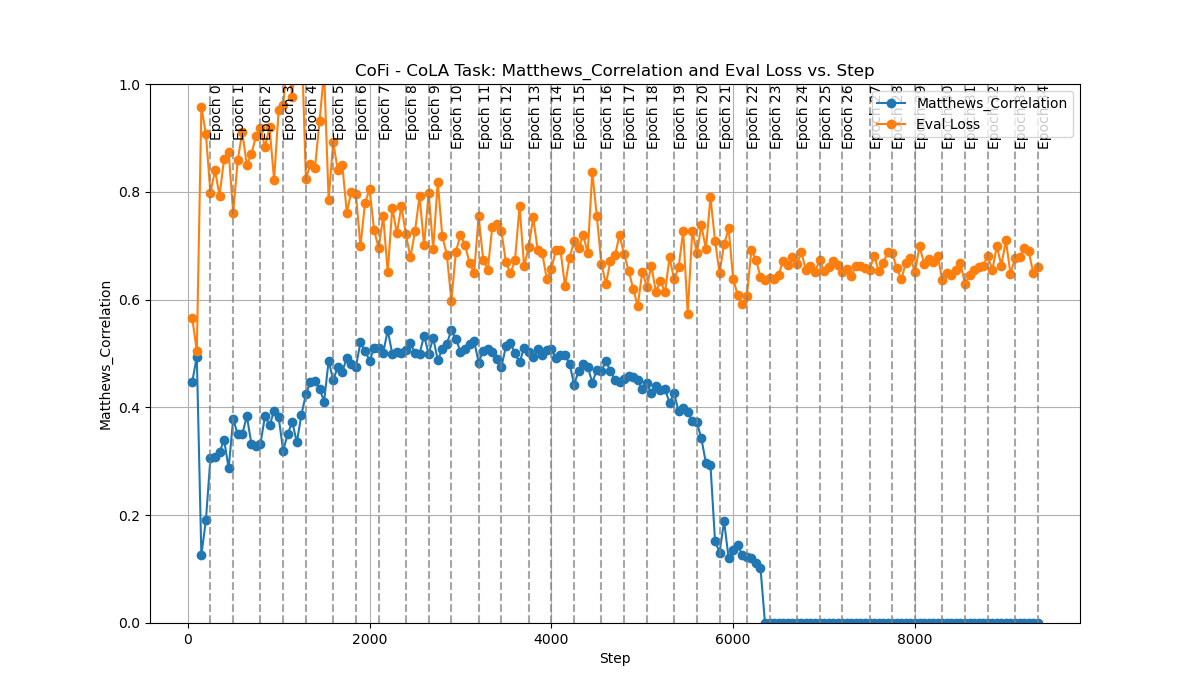


*Figure 2: Accuracy & Evaluation Loss vs. Step Graph for CoFi - MNLI Task*

One thing to note, on the first couple of steps we see our accuracy start off very low and our eval loss start out really high. We believe this is an issue with our model logging file. We noticed that our model accuracy increases substantially in the first couple of steps getting around 0.8 before leveling off. It then remains steady until step 20000 in which it starts decreasing. Our evaluation loss shows a similar story: it begins to stop decreasing until step 2500 and levels off, it has more variance compared to the accuracy data. It then starts increasing again around step 20000. The model then appears to perform worse around epoch 1 taking massive dips in accuracy that stop around step 26000 and start slowly increasing. Eval loss shows a similar story increasing around epoch 1 and then stopping around step 26000 before decreasing again. With that in mind we ran the training again and took the model from epoch 0 before the training accuracy decreased and the evaluation loss increased.

### ***CoFi - CoLA***

The second test we ran with CoFi was CoLA, we used another fine tuned model called: [bert-base-uncased-finetuned-cola](https://huggingface.co/senfu/bert-base-uncased-finetuned-cola). The same CoFi parameters were used. Results are below.



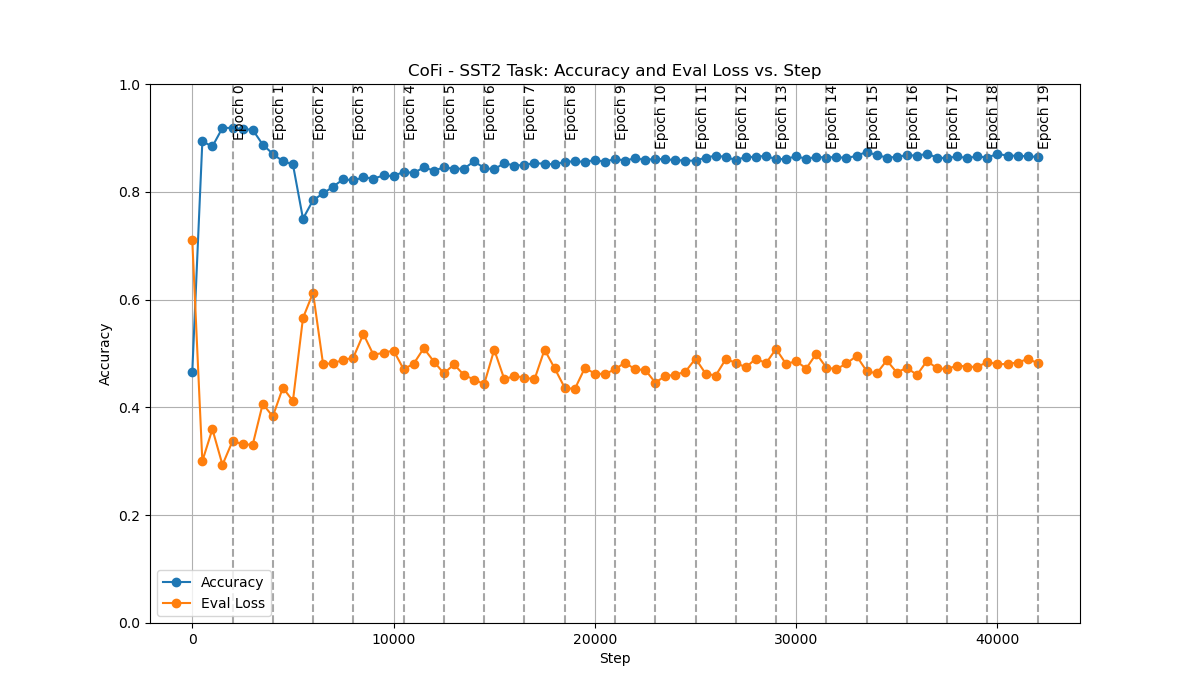
*Figure 3: Matthews Correlation & Evaluation Loss vs. Step Graph for CoFi - CoLA Task*

At the start of the CoLA task we see that the Matthews\_Correlation and the evaluation loss are close together and they then separate sharply. This mirrors what we saw in our MNLI task which makes us believe it has to do with the logging function. Matthews\_Correlation increases up until approximately epoch 6 before it levels off and starts decreasing. When it hits epoch 20 it sharply decreases until it hits 0. Our team does not have as much experience with Matthews correlation, so we don’t know if this is expected or an anomaly. We see that evaluation loss has a lot of variance between data points jumping up and down with the point that has the lowest difference between Matthew correlation and evaluation loss being epoch 10. We run the task again and take the model at that point for further analysis

### 

### *CoFi -* ***SST-2***

Finally we ran the same script for the SST-2 task. We used a BERT model that has been fine tuned on the SST-2 task called [bert-base-uncased-finetuned-sst2](https://huggingface.co/senfu/bert-base-uncased-finetuned-sst2). The same parameters of the previous tasks were used



*Figure 4: Accuracy & Evaluation Loss vs. Step Graph for CoFi - SST-2 Task*

For the SST-2 task the same starting pattern that the other models exhibited was present. There is a sharp increase in the accuracy that levels of at epoch zero and slightly decreases until epoch 2, where the model then bounces back up and levels of around 0.85 accuracy. The evaluation loss shows the same story but inverted with more variance between data points than the accuracy and reaching its maximum around epoch 2, with the accuracy getting its minimum at the same time. We take the model at Epoch 0 since it has the highest accuracy and the lowest evaluation loss.

**Results Conclusion**

After we ran the model training we benchmarked the models we created with another set of models that the creators of CoFi provided. We ran our model training one more time after the above graphs and stopped it at a certain epoch, which we specified. We then evaluated the model on tasks, getting how long it took the model to infer an example. We then compared three models, using bert-base-uncased as the reference model compared with our models and the creators of CoFi (denoted by the princeton-nlp prefix) on accuracy and speedup.

Speedup rate is the measurement we use as the compression rate does not necessarily reflect the actual improvement in inference latency. We use the unpruned BERT base model as the baseline and evaluate all the models with the same hardware setup on a NVIDIA Tesla P100 GPU to measure inference speedup. The input size is 128 for GLUE tasks and we use a batch size of 128. The table of our results is listed below:

| **MNLI Tasks** |  |  |
| --- | --- | --- |
| Model Name | Accuracy | Speedup |
| bert-base-uncased | 0.5103 | 1 |
| princeton-nlp/CoFi-MNLI-s95 | 0.8054 | 10.29575071 |
| OurModelMNLI | 0.8182 | 12.36190476 |
| **SST2 Tasks** | Accuracy | Speedup |
| bert-base-uncased | 0.6052 | 1 |
| princeton-nlp/CoFi-SST2-s95 | 0.9037 | 10.4310344 |
| OurModelSST2 | 0.8658 | 8.9408867 |
| **CoLA Tasks** | Matthew Correlation Multiplier | Speedup |
| bert-base-uncased | 1 | 1 |
| princeton-nlp/CoFi-CoLA-s95 | 1.573264781 | 9.593175853 |
| OurModelCoLA | 1.597911227 | 9.092039801 |

*Table 1: Results and Comparison of Models on GLUE Tasks*

The bert-base-uncased model is a 1 since it is then compared to the other models. For the MNLI task we see that our model achieved an accuracy of 0.8182 compared to an accuracy of 0.8054. The speedup was similar as well with 10.3 and 12.36 respectively having our model slightly beat the creators in that regard.

The SST2 task was a different story with the princeton-nlp model beating our model in accuracy 0.9037 compared to 0.8658 and having a higher speedup 10.43 compared to 8.94.

Finally in the CoLA tasks we see that the princeton-nlp model performed better (lower is better for Matthew correlation) at 1.57 compared to 1.60 and a speedup of 9.59 vs 9.09.

Overall our model results validate the ones that the CoFi creators saw and we achieved similar results.

**Conclusions and Future Work**

The aim of our project was to verify the effectiveness of the CoFi pruning method by implementing our own models and comparing them with known baselines. We showed that CoFi does in fact lead to significant inference time speed up, sometimes in the case of double digits, with little drop in accuracy. We showed this by comparing a base BERT model with the author's models and verifying the work by using our own models.

We understand that our model’s were not the best, showing symptoms of bad model training. However we would like to point out the aim of our project was to show that CoFi works without considering the generalizability of our project. We would have liked to have performed a hyperparameter search to improve our model training. But we had numerous limitations that prevented us from doing such a thing. Firstly, we are using Google Cloud Compute Engine. We did not have access to a computer with CUDA. They originally tried adopting the training algorithm to work on the MPS backend (Edman’s computer had access to that) but it ultimately failed. We then decided to work on a cloud service that had CUDA devices, giving us access to faster GPUs than we were capable of acquiring ourselves. This was the first time that either of us had ever used a cloud service and we had significant learning hurdles to overcome. Some examples include: Using SSH to access the server, setting up billing with the provider, setting up the VM (which took numerous hours due to issues with the providers). In the end we were able to solve most of these issues but not without exhausting multiple weeks of troubleshooting, before we even had the chance to run our models.

While we would have liked to perform further hyperparameter tuning of our model/tests we found out that we had run into the credit usage of our cloud service provider. We are still trying to sort that out but currently we have a 100$ negative balance in our account. Because of this we currently do not have access to our VM that has the model logs/training data. This entire report was made with an older version of Logs that we had a local copy off. Overall this project was a massive learning experience for us in terms of building our own models, tuning them, using a cloud service provider, and learning how valuable it is to have a local copy.

In terms of project management Edman was tasked with implementing and setting up the VM, Cassandra was in charge of reading the research papers and picking the algorithm to use

**References**

* <https://dl.acm.org/doi/pdf/10.1145/3580305.3599284>
* <http://proceedings.mlr.press/v119/goyal20a/goyal20a.pdf>
* <https://arxiv.org/pdf/2107.00910.pdf>
* <https://aclanthology.org/2022.acl-long.502.pdf>
* <https://github.com/princeton-nlp/CoFiPruning>