IT-UNIVERSITETET I KØBENHAVN

Part 2 : BERT - Pruning

BERT, CoFi Pruning, Distillation, L0 regularisation

Ambroise LAROYE–LANGOUËT

June 27, 2024

Contents

1 Introduction / context

In my previous work, I studied two types of pruning: Magnitudes and Wanda, on the Llama model. Initially, I did not find satisfactory results regarding the correlation between energy consumption and the sparsity ratio. I then studied structured pruning.
In this new section, I propose to study pruning on the BERT model. My approach to pruning is completely different. I no longer study the model independently, but instead, the goal is to adapt the weights gradually during training (structured pruning). And the idea is to improved this pruning, with external parameters, configuration (distillation, adapters, fine tuning).

2 BERT

Bidirectional Encoder Representations from Transformers[1]

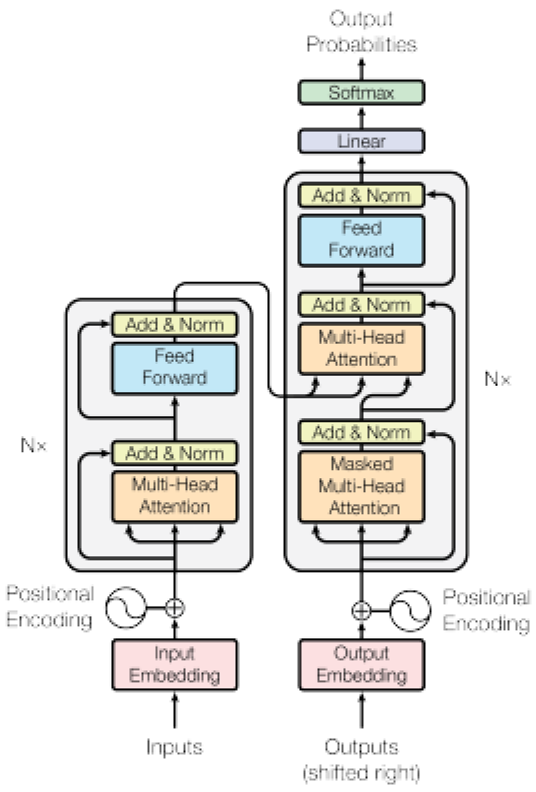


Figure 1: BERT : Transformer - model architecture

We can divide this model into 2 parts: decoding and encoding. And I'm going to go deeper into the block principles. Multi-Head Attention, Position Encoding and Feed Forward.
BERT is composed of 12 layers. We can resume this model with the following simplified

[1]`https://arxiv.org/abs/1706.03762`

schematic[2] :

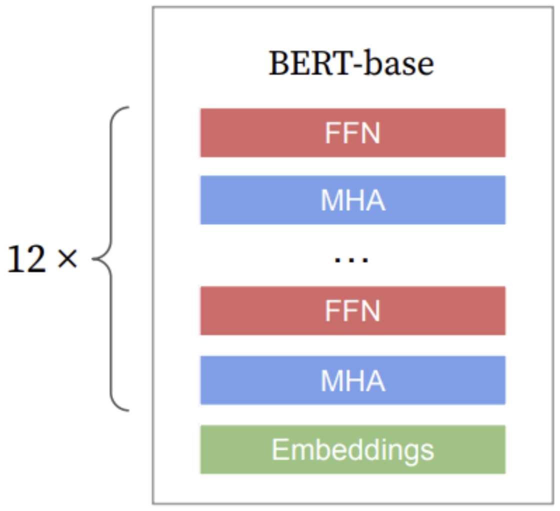


Figure 2: BERT - simplified architecture

2.1 Positional Encoding

It inject some information about the relative or absolute position of the tokens in the sequence.
Using cosine functions :

$$PE(pos, 2i) = sin\left(\frac{pos}{pos10000^{2i/d_{model}}}\right)$$

$$PE(pos, 2i) = cos\left(\frac{pos}{pos10000^{2i/d_{model}}}\right)$$

These are playing the role of embedding in this model.

[2]`https://arxiv.org/abs/2204.00408`

2.2 Multihead Attention (MHA)

The Multi-Head Attention is split in 3 projections : Query, Key and Value. The operations carried out on these layers are shown in the diagram below.
The initial layer is divied in $\frac{768 \text{ (model size)}}{h}$ parallel attention layers. The aim of this operation is to reduce the computation cost.

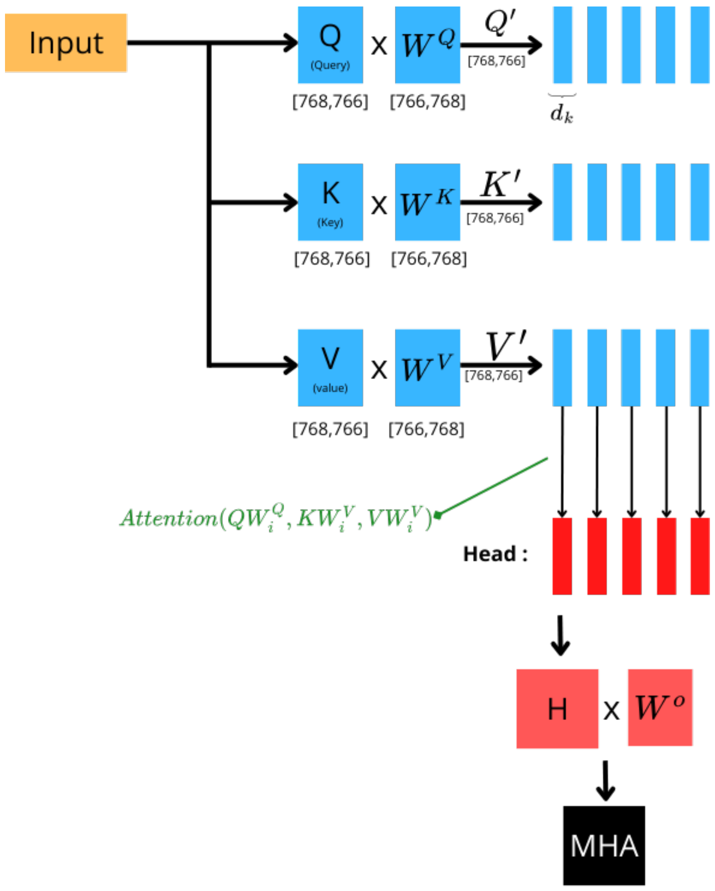


Figure 3: Multihead Attention on the layers Q, K and V

1.
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

with

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

2. The concatenated output is then projected using a final weight matrix W^O:

$$\text{MHA}(x) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

2.3 Feed Forward Networks : FFN

This consists of two linear transformations with a ReLU activation in between.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

3 Objectives and suggestions

The general idea is to add distillation just before the pruning.
The aim is to reduce the model as much as possible. And see if we lose accuracy. And we'll explore energy costs, model size and speedup.
So we use the BERT model : google-bert/bert-base-uncased[3]
This time, we will adjust the model weights dynamically based on the loss, and we will train and evaluate all our results on a specific task that we will choose from GLUE[4]. We'll be back on GLUE
Previously we have seen that, MHA is a function of $(W_i^Q, W_i^K, W_i^V, W_o^i, X)$
The idea is to add the parameter z_{head}^i in MHA :

$$MHA(X) = z_{head}^i * Concat(head_1, ...head_h)W^o$$

with z corresponding to the weight of the pruning mask.

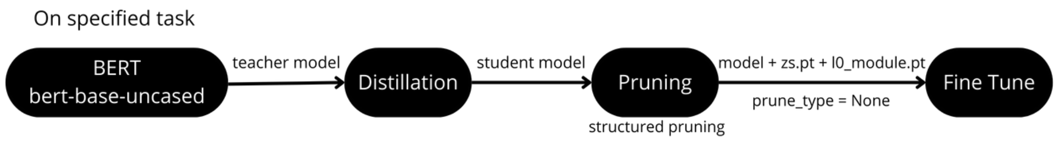


Figure 4: Step of the algorithm

$$\max_{\lambda_1,\lambda_2} \min_{\theta,\hat{s}} \left(\lambda \mathscr{L}_{pred}(\theta) + (1-\lambda)\mathscr{L}_{layer}(\theta) + \lambda_1(s^* - \hat{s}) + \lambda_2(s^* - \hat{s})^2\right)$$

[3]https://huggingface.co/google-bert/bert-base-uncased
[4]https://arxiv.org/abs/1804.07461

with :

$$\begin{cases} s^* := \text{ the target sparsity} \\ \hat{s} := \text{ the expected sparsity} \\ \mathscr{L}_{layer} \text{ and } \mathscr{L}_{pred} \text{ are explained later} \end{cases}$$

4 CoFi (Coarse and Fine grained) Pruning

4.1 Distillation

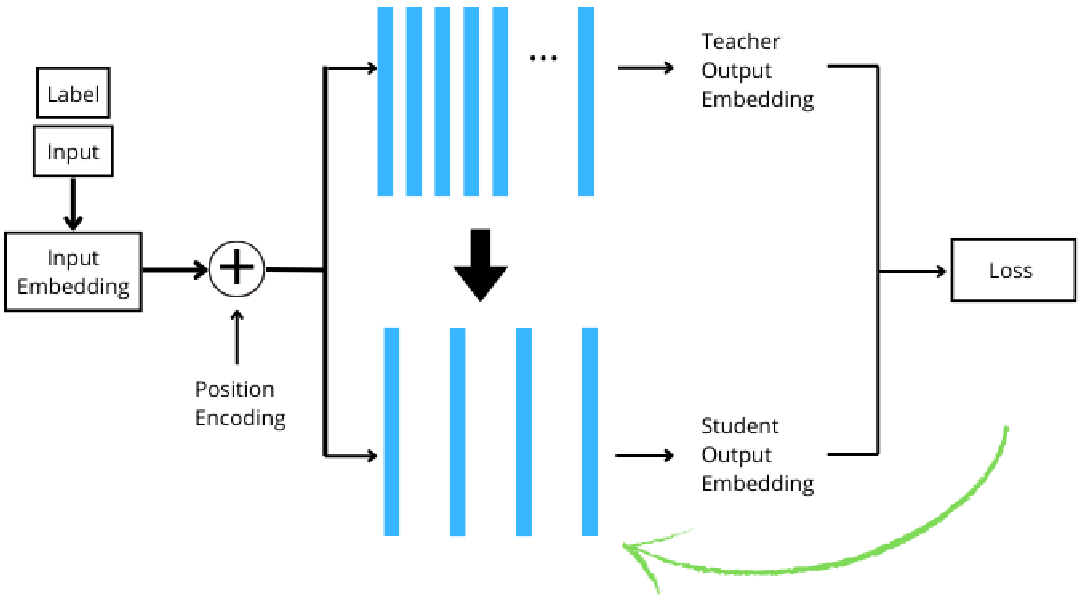


Figure 5: Algortihme Distillation Diagram

In the diagram above, I illustrate how distillation works. We take the initial model, keeping only 4 layers chosen arbitrarily. The objective is then to complete the weights using backpropagation of the loss on the so-called student model. This should allow us to reduce the model for the first time.

4.2 Pruning

4.2.1 L0 Regularisation

L0 regularization aims to create a pruning mask that helps in selecting which weights to retain, thus enforcing sparsity. Here's a short explanation of the process:

Pruning Mask Creation : The first step is to create a pruning mask, which will be converted into a tensor. This mask contains the coordinates of the weights that should be retained during training.
Lagrange Regularization : The mask is updated continuously during the training process using Lagrange regularization. This involves two lambda parameters, λ_1 and λ_2, which control the sparsity level. These parameters are adapted and optimized during training to find the optimal by using `zero_grad()` optimizer from PyTorch.
The tensor `zs`, which represents the weights, is divided into three parts: `head_z`, `intermediate_z`, and `hidden_z`. This division helps in managing different sections of the network independently, allowing for more fine-grained control over the pruning process.
Index List : Based on the updated pruning mask, an index list is created. This list contains the indices of all the weights that are to be retained after pruning.

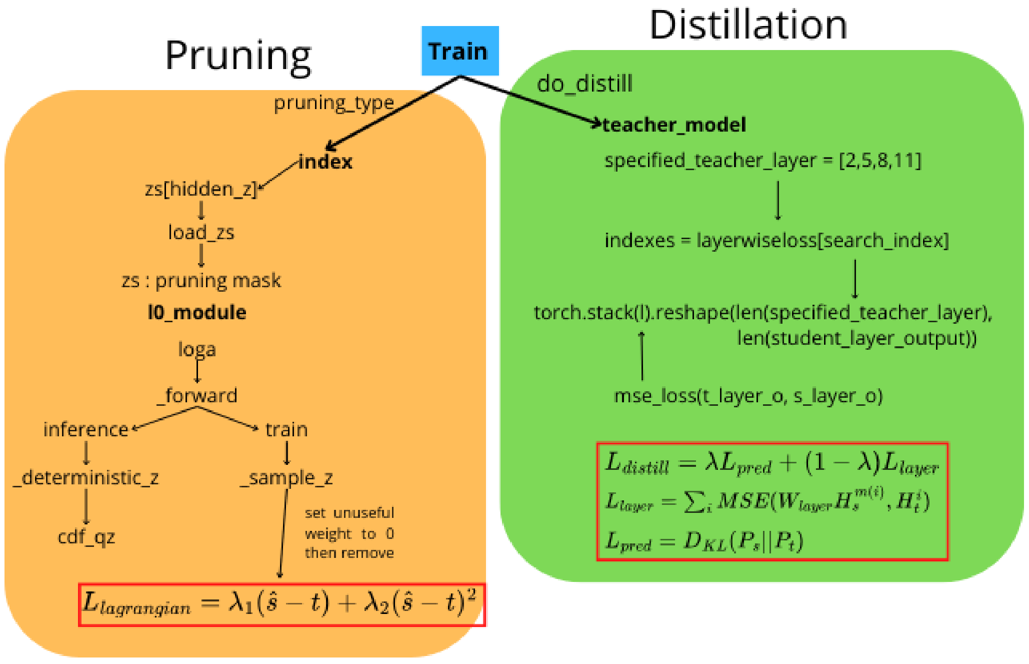

Pruning
Train
Distillation
pruning_type
index
zs[hidden_z]
load_zs
zs : pruning mask
l0_module
loga
_forward
inference
train
_deterministic_z
_sample_z
cdf_qz
set unuseful weight to 0 then remove
$L_{lagrangian} = \lambda_1(\hat{s} - t) + \lambda_2(\hat{s} - t)^2$
do_distill
teacher_model
specified_teacher_layer = [2,5,8,11]
indexes = layerwiseloss[search_index]
torch.stack(l).reshape(len(specified_teacher_layer), len(student_layer_output))
mse_loss(t_layer_o, s_layer_o)
$L_{distill} = \lambda L_{pred} + (1 - \lambda)L_{layer}$
$L_{layer} = \sum_i MSE(W_{layer}H_s^{m(i)}, H_t^i)$
$L_{pred} = D_{KL}(P_s||P_t)$

Figure 6: Training process through distillation and pruning

4.3 Fine Tuning

Fine-tuning is often used with BERT. It allows adaptation to specific tasks and improves performance by adjusting the weights of the pre-trained model on a specific dataset.
Pre-training models like BERT requires massive amounts of data and computational resources. Fine-tuning, on the other hand, can be done with much smaller datasets because the model has already learned a rich representation of the language. Thus, it is more ac-

cessible and less costly in terms of data and computation.
Reduced development time: Fine-tuning a pre-trained model is much faster than training a model from scratch.

4.4 Evaluation - Results

4.4.1 Tasks

GLUE[5] : General Language Understanding Evaluation. GLUE is a batch of task like MNLI (Multi-Genre Natural Language Inference Corpus (Williams et al., 2018) is a crowd-sourced collection of sentence pairs with textual entailment annotations), CoLA (Corpus of Linguistic Acceptability), MRPC (Microsoft Research Paraphrase Corpus is a corpus of sentence pairs automatically extracted from online news sources), QNLI (Stanford Question Answering Dataset is a question-answering dataset consisting of question-paragraph pairs)...

- Inference tasks
  - MNLI : Multi-Genre Natural Language Inference. Predict whether the premise entails the hypothesis.
  - QNLI : This task is to determine whether the context sentence contains the answer to the question
  - RTE : Recognizing Textual Entailment. Predict whether a given piece of text (the hypothesis) can be inferred or logically follows from another piece of text (the premise).
- Single sentence tasks
  - CoLA : Corpus of Linguistic Acceptability. Each example is a sequence of words annotated with whether it is a grammatical English sentence.
  - SST-2 : Stanford Sentiment Treebank. Predict the sentiment of a given sentence
- Similarity and paraphrase tasks

[5]`https://arxiv.org/abs/1804.07461`

– MRPC : Microsoft Research Paraphrase Corpus. Predict whether the sentences in the pair are semantically equivalent

– QQP : e Quora Question Pairs. Determine whether a pair of questions are semantically equivalent. Evaluation report also F1 score.

4.4.2 Model size

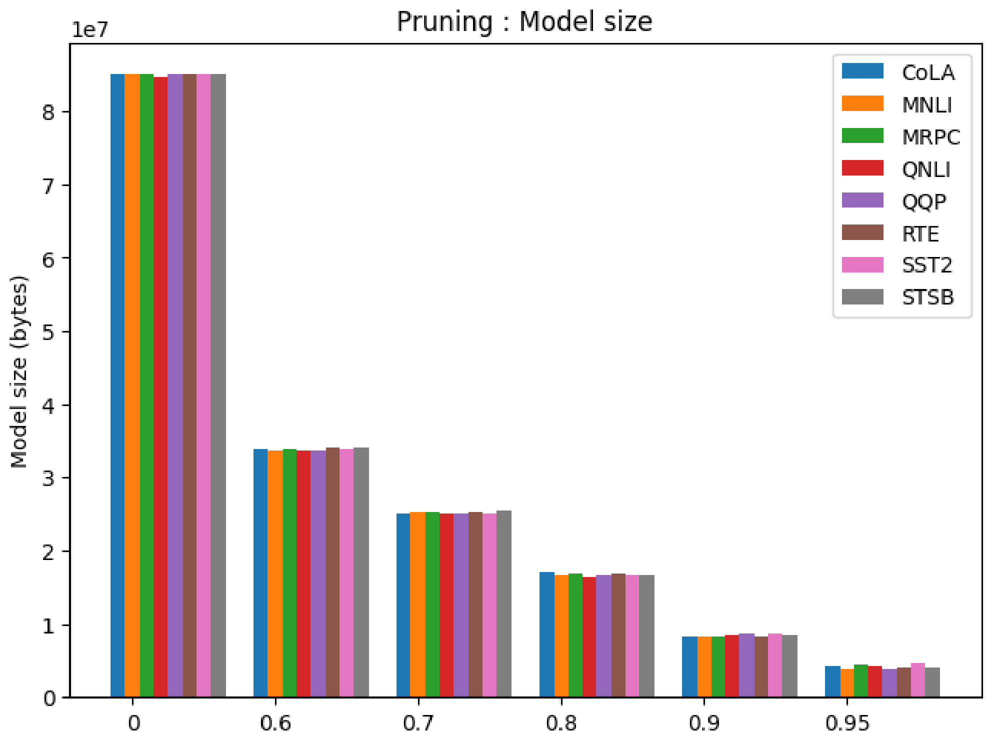


Figure 7: Reduction of the model size

We're well on our way to reducing the model. Keeping accuracy more or less constant. However, the accuracy is very low. So we're going to have to supplement this with fine tuning.
In the table below, accuracy is represented **before the second fine tuning** as a function of task.

Task	MNLI	QQP	QNLI	SST2	MRPC	RTE
Accuracy	0.355	0.6318	0.4946	0.4885	0.3162	0.5271

And **after re-fine tune** the accuracy has increase.

Task	MNLI	QQP	QNLI	SST2	MRPC	RTE
Accuracy	0.8195	0.9062	0.8948	0.9048	0.8431	0.6462

Note:
Please note that in my work, I refer to fine-tuning after pruning. However, there are two cases of fine-tuning. The first occurs during pruning, where the model training adjusts and removes weights over 20 epochs for the tasks: MNLI, QQP, QNLI, and SST2, and 100 epochs for the tasks: MRPC, RTE, STSB, and CoLA.
Since the accuracy is too low, we then perform only fine-tuning on the pruned model weights. This time, we run 20 epochs for all tasks. Hence the term "re-fine-tuning."

4.4.3 Emissions

First of all, I didn't measure whether the distillation and pruning process actually saved energy. But I did evaluate the whole process. By measuring energy throughout training and distillation. We report these results in the graph below. I also compared the results according to the sparsity ratio (0, 0.6, 0.7, 0.8, 0.9, 0.95).

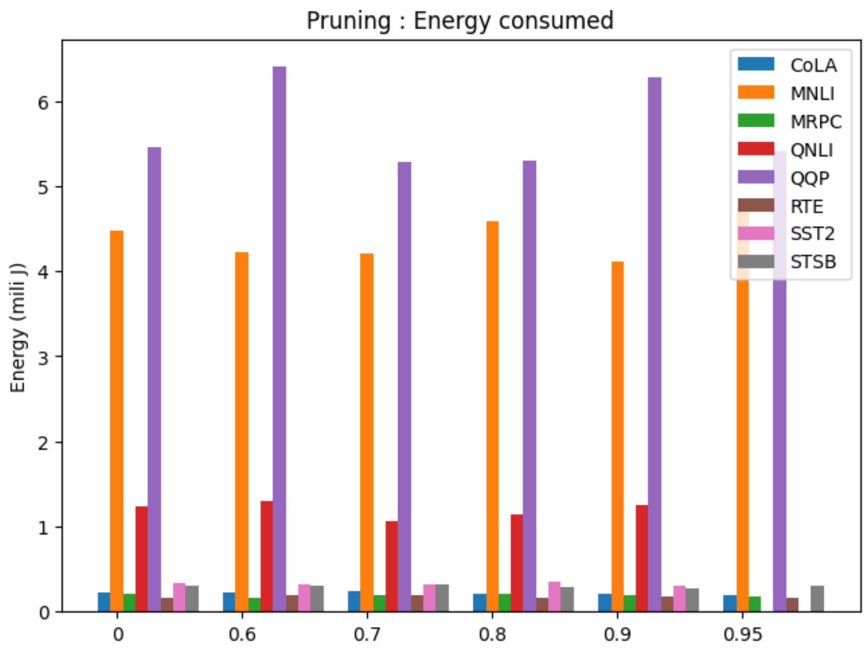


Figure 8: Energy consumed by an inference on each task

I can now compare whether a task is energy-consuming. We can see that the QQP, MNLI and QNLI models consume the most energy during pruning.

Pruning speed up the training process, that should involves a reduction of energy. Let's look at the inference time and energy cost.

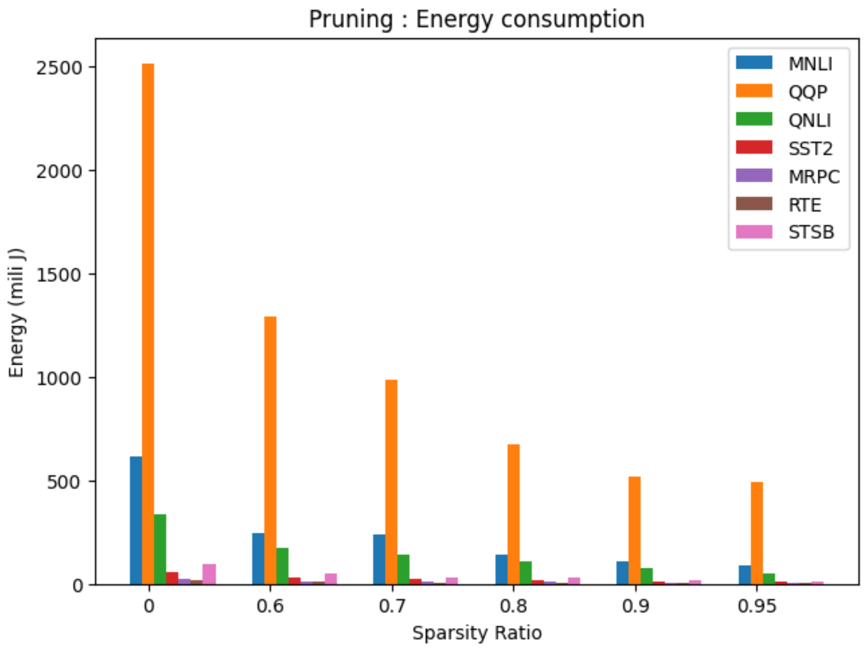


Figure 9: Energy consumed by an inference on each task

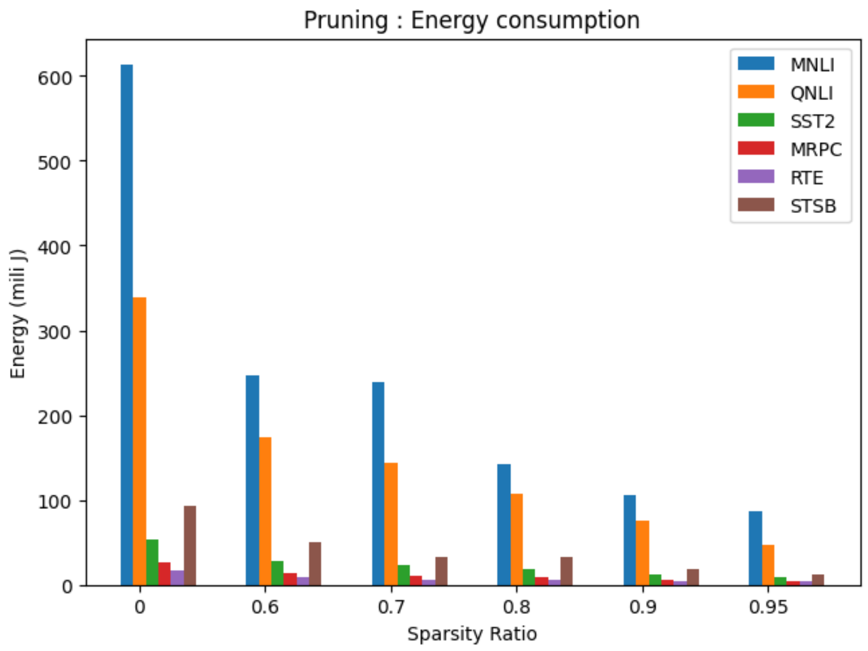


Figure 10: Energy consumed by an inference on each task (zoom)

Comparison between the emissions and the model size according to the sparsity target.

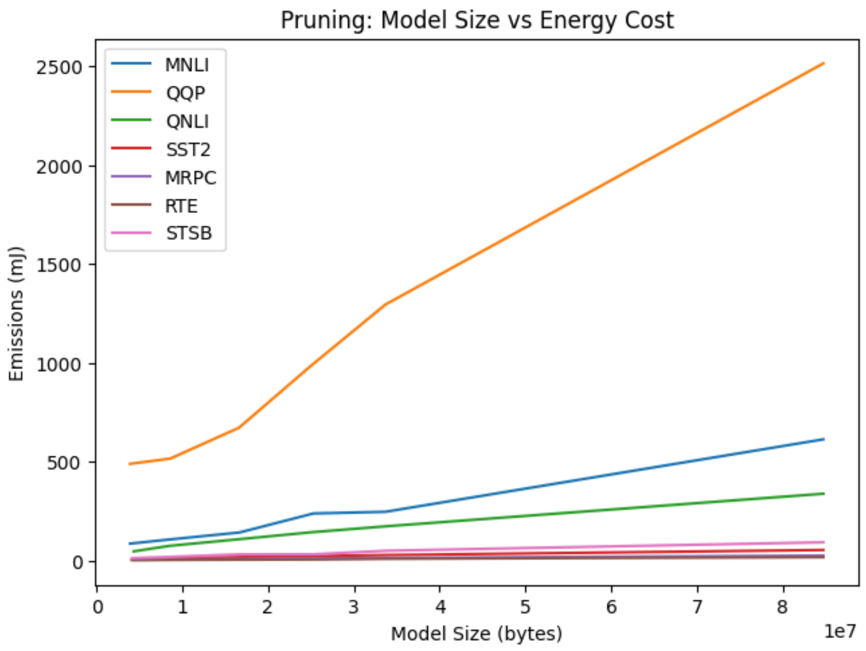


Figure 11: Energy and model size for each sparsity

4.4.4 Speedup

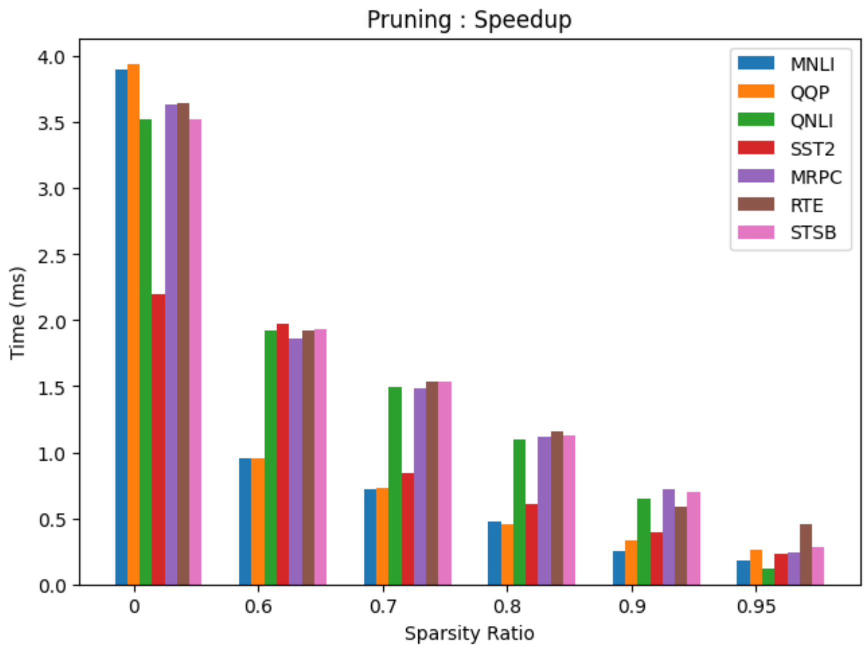


Figure 12: Inference time of each task depending on the sparsity

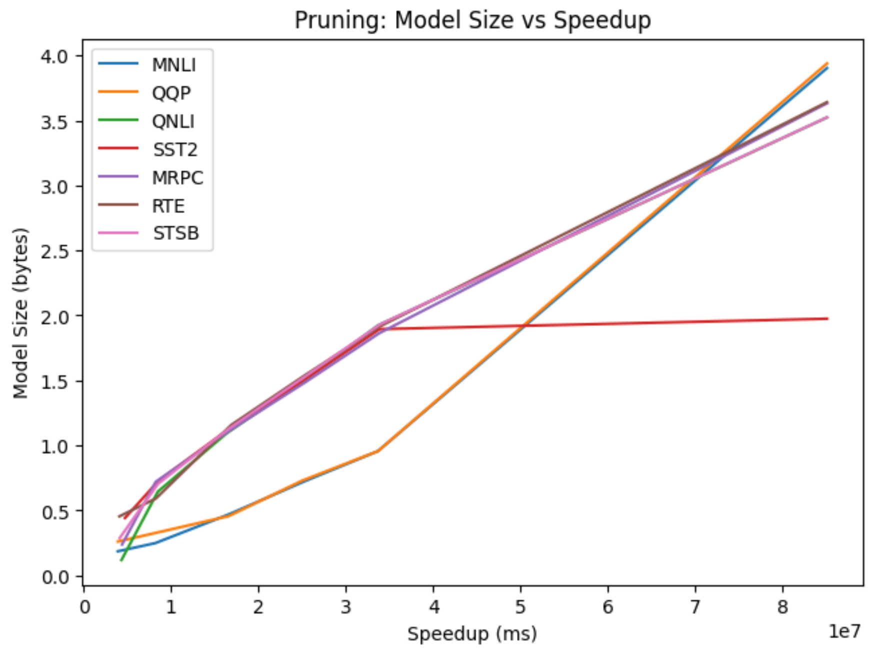
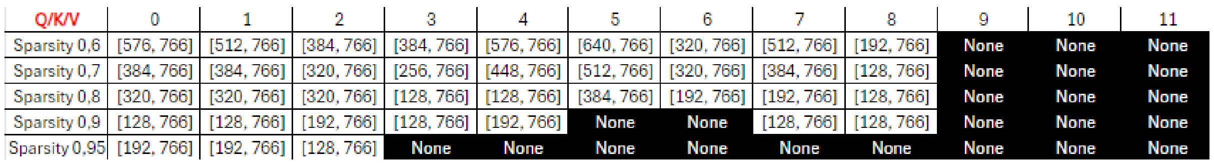


Figure 13: Speedup and model size

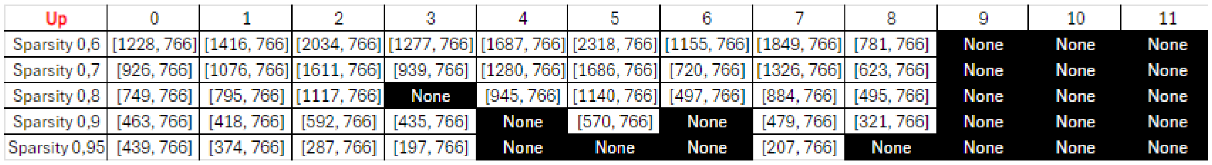
4.4.5 Layers

3 other layers that haven't been explore in the explanation of BERT architecture : Up layer, Down layer and Output layer. First, let's note that $size(Query) = size(Key) = size(Value) = transpose(Output)$ and $size(Down) = transpose(Up)$
By observing the layers during pruning and as a function of sparsity we notice several elements:
Layers are annotated from 0 to 11 (so 12 in total).
At a sparsity of 0.6, layers 9, 10 and 11 are removed. Then layer sizes are modified according to sparsity. And some layers are removed in one layer but reappear in the next. There's no apparent trivial logic. Because the model only adapts to the task in hand.
In the table below, I've plotted the matrix sizes of the layers.



Q/K/V	0	1	2	3	4	5	6	7	8	9	10	11
Sparsity 0,6	[576, 766]	[512, 766]	[384, 766]	[384, 766]	[576, 766]	[640, 766]	[320, 766]	[512, 766]	[192, 766]	None	None	None
Sparsity 0,7	[384, 766]	[384, 766]	[320, 766]	[256, 766]	[448, 766]	[512, 766]	[320, 766]	[384, 766]	[128, 766]	None	None	None
Sparsity 0,8	[320, 766]	[320, 766]	[320, 766]	[128, 766]	[128, 766]	[384, 766]	[192, 766]	[192, 766]	[128, 766]	None	None	None
Sparsity 0,9	[128, 766]	[128, 766]	[192, 766]	[128, 766]	[192, 766]	None	None	[128, 766]	[128, 766]	None	None	None
Sparsity 0,95	[192, 766]	[192, 766]	[128, 766]	None	None	None	None	None	None	None	None	None

Figure 14: Layers size of Query, Key and Value



Up	0	1	2	3	4	5	6	7	8	9	10	11
Sparsity 0,6	[1228, 766]	[1416, 766]	[2034, 766]	[1277, 766]	[1687, 766]	[2318, 766]	[1155, 766]	[1849, 766]	[781, 766]	None	None	None
Sparsity 0,7	[926, 766]	[1076, 766]	[1611, 766]	[939, 766]	[1280, 766]	[1686, 766]	[720, 766]	[1326, 766]	[623, 766]	None	None	None
Sparsity 0,8	[749, 766]	[795, 766]	[1117, 766]	None	[945, 766]	[1140, 766]	[497, 766]	[884, 766]	[495, 766]	None	None	None
Sparsity 0,9	[463, 766]	[418, 766]	[592, 766]	[435, 766]	None	[570, 766]	None	[479, 766]	[321, 766]	None	None	None
Sparsity 0,95	[439, 766]	[374, 766]	[287, 766]	[197, 766]	None	None	None	[207, 766]	None	None	None	None

Figure 15: Layers size of Up layer

Question that can arise :
We note that the first layers are not the ones that have been prune the most. Are they essential to ensure the model's perplexity? Do they carry the main information?

4.4.6 Loss

In this part, I found it interesting to observe the evolution of hyperparameters like λ_1 and λ_2, which are used in the calculation of the Lagrangian loss.
These hyperparameters are compute with the optimizer Adam.
Let's look at the variation of λ_1 and λ_2 that allow the lagrangian regularisation.

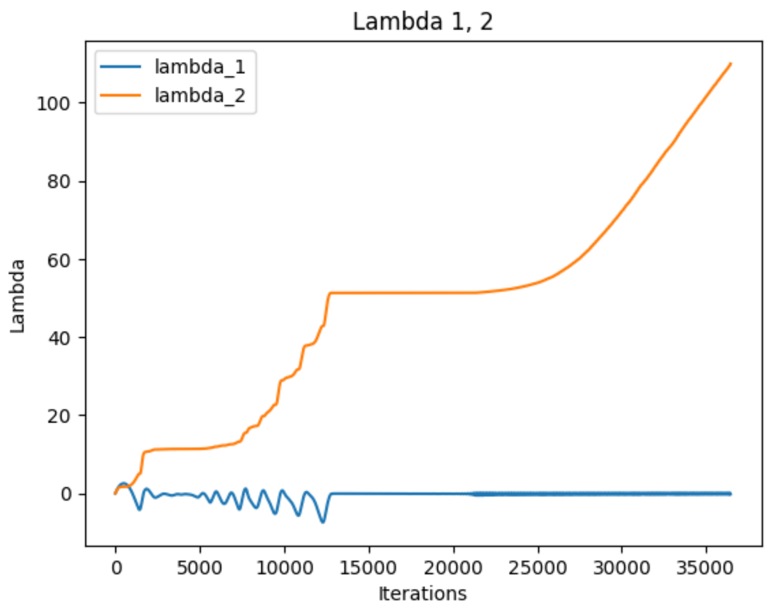


Figure 16: Evolution of λ_1 and λ_2 during iterations

We notice that, over the iterations, λ_2 (regulating the term : $(\hat{s} - t)^2$) skyrockets, while λ_1 stabilizes around 0, making the term $(\hat{s} - t)$ negligible. As λ_2 increases significantly, the term $(\hat{s} - t)^2$ dominates the loss function. The optimizer is heavily penalizing the squared difference, forcing $\hat{s}$ to be very close to t.

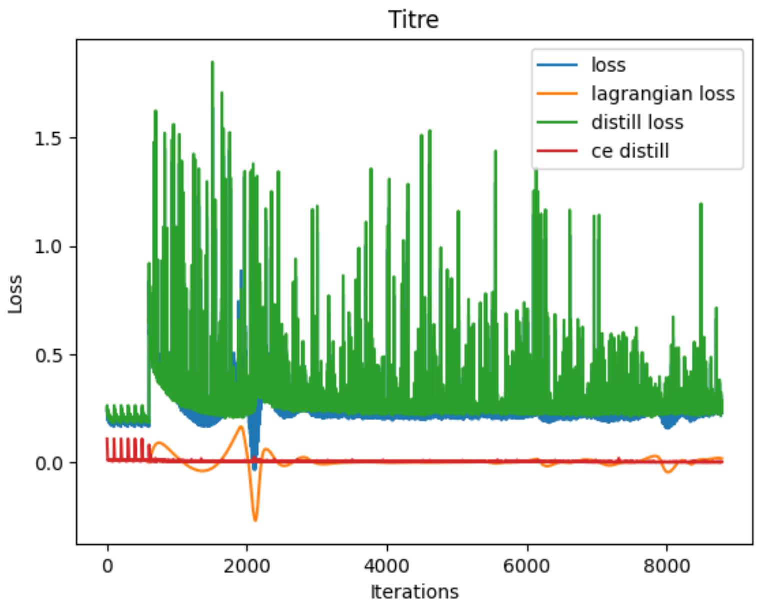


Figure 17: Evolution of the different *Loss* function during iterations (distillation loss and pruning loss)

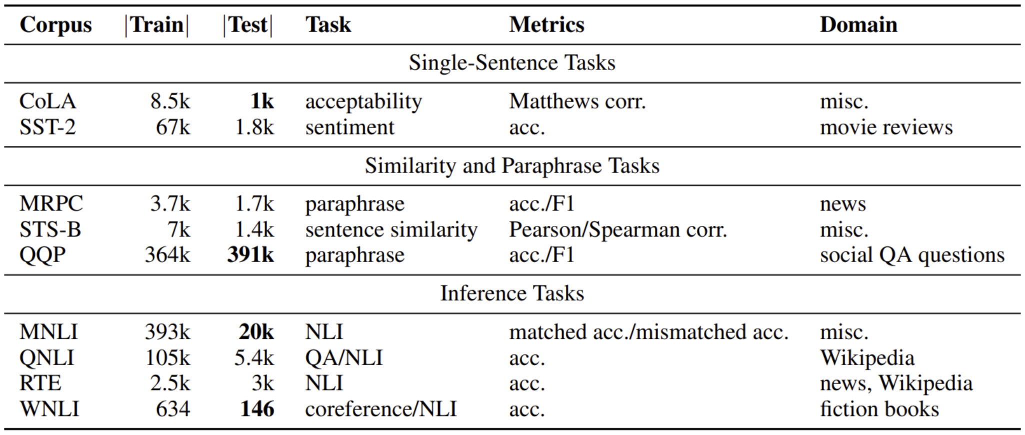
4.5 Benefits

In this section, I also refer to the previous report I conducted on unstructured pruning, using the Llama model on Wanda and magnitude pruning.
Thanks to structured pruning, we achieve the desired results through pruning. We have greatly reduced the model size and energy consumption while maintaining almost the same accuracy as we had without pruning.
The use of pruning and the sparsity rate depends on what we aim to prioritize: speedup, model size, accuracy, or reducing energy cost.
We have seen that distillation beforehand can be potentially double-edged, because although we further reduce the model, it is necessary to retrain the model afterward. In other words, distillation requires more iterations.

4.6 Weaknesses and Improvements

We can identify some weaknesses in this work. In this section, I highlight elements that could be improved, limitations, and points that need further work. In the following sections, I will address and work on these points of improvement.

GLUE : How to understand these disparity between tasks
In our results, we observed certain disparities between the tasks. Some tasks are slower and thus more energy-consuming than others. Only the model size does not differ significantly between the tasks. The main reason for this disparity is likely the number of parameters. We can support this argument with the following table, derived from the article : *GLUE: A MULTI-TASK BENCHMARK AND ANALYSIS PLATFORM FOR NATURAL LANGUAGE UNDERSTANDING* [6]



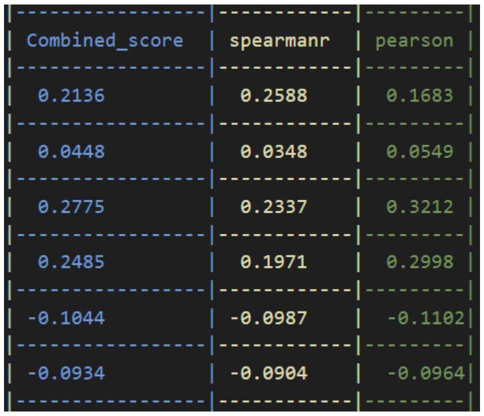
Corpus	|Train|	|Test|	Task	Metrics	Domain
			Single-Sentence Tasks		
CoLA	8.5k	**1k**	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
			Similarity and Paraphrase Tasks		
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	**391k**	paraphrase	acc./F1	social QA questions
			Inference Tasks		
MNLI	393k	**20k**	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	**146**	coreference/NLI	acc.	fiction books

Figure 18: GLUE task descriptions and statistics. All tasks are single sentence or sentence pair classification, except STS-B, which is a regression task

The main limitation of using GLUE is that all its tasks are binary. This means that the tasks only involve responding in a negative or positive manner. There are only two possible outcomes, resulting in a 50% chance of being incorrect. This introduces a significant element of randomness, potentially leading to a less accurate model.
In GLUE task, only STS-B is a regression task, our evaluations parameters are, combined score, spearmanr (Measures the rank-order correlation) and pearson (Measures the linear

[6] `https://arxiv.org/abs/1804.07461`

correlation between two sets of data). These data are in the range -1, 1
STS-B[7] **:** The Semantic Textual Similarity Benchmark is a collection of sentence pairs drawn from news headlines, video and image captions, and natural language inference data.
Our results on STS-B task, without a re-train of the model are :



```
|-----------------|------------|---------|
| Combined_score  | spearmanr  | pearson |
|-----------------|------------|---------|
|  0.2136         |  0.2588    |  0.1683 |
|-----------------|------------|---------|
|  0.0448         |  0.0348    |  0.0549 |
|-----------------|------------|---------|
|  0.2775         |  0.2337    |  0.3212 |
|-----------------|------------|---------|
|  0.2485         |  0.1971    |  0.2998 |
|-----------------|------------|---------|
| -0.1044         | -0.0987    |  -0.1102|
|-----------------|------------|---------|
| -0.0934         | -0.0904    |  -0.0964|
|-----------------|------------|---------|
```

Figure 19: Evaluation of BERT model on STS-B task

The results obtained are not very satisfactory. We do see a decrease in the combined score as a function of the sparsity ratio. It seems that the model is less accurate for slightly more complex tasks. Beyond a sparsity ratio of 0.8, the model becomes unreliable (considering that a result below 0 is regarded as a random model).

Additionally, we have evaluated the model solely on the GLUE tasks. It would be prudent to evaluate it on a wider range of tasks to ensure generalization and not limit ourselves to "simple" tasks. This way, we can observe if there are any divergences in the results.

Distillation
As mentioned earlier, model distillation combined with pruning can negatively impact accuracy and thus require a greater number of iterations. It might then be interesting to compare results with and without distillation, and to vary the initialization of the student

[7]`https://arxiv.org/abs/1804.07461`

model.
In the previous part, we used only 4 out of 12 layers (i.e., 1/3 of the model's layers), whereas many articles recommend retaining 50% of the model.[8] [9]
Fine tuning An alternative to fine-tuning our model exists: adapters. This approach is less resource-intensive and allows for better adaptation of the model to numerous tasks.

5 Without distillation

5.1 Accuracy

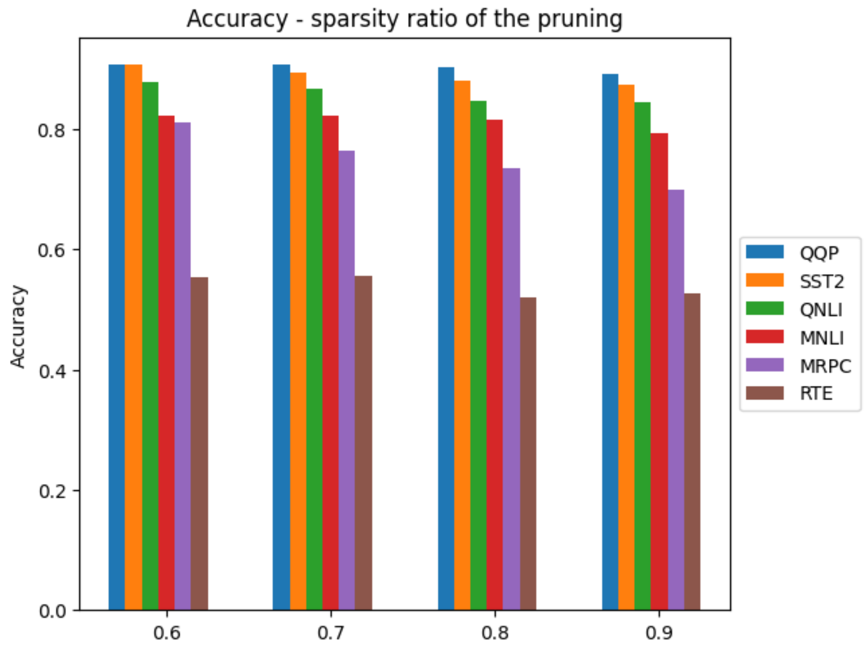


Figure 20: Accuracy of each task depending on the sparsity from 0 to 0.9

[8]`https://arxiv.org/pdf/1909.10351`
[9]`https://arxiv.org/pdf/1910.01108`

Overall, we are abserving a slithly decrease of the accuracy from the sparsity 0.6 to 0.9. It seems that QQP and SST2 tasks are more accurate and relevant for the BERT model compared to RTE. There is no need to refine-tune after pruning; the accuracy is already sufficient. At the moment, it doesn't appear that there is much to gain from distillation before pruning.

5.2 Model size

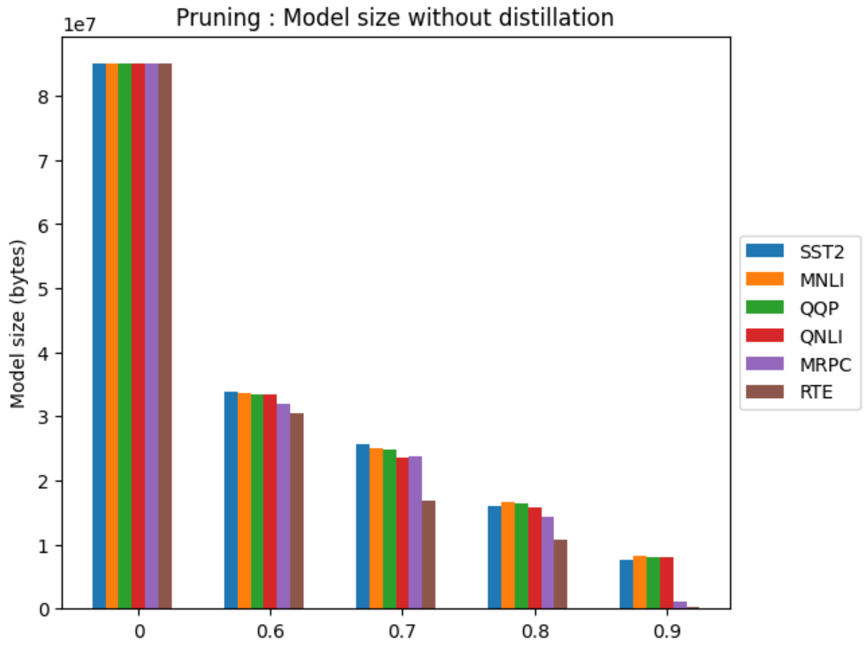


Figure 21: Model size of each task depending on the sparsity

We can draw a parallel with what we observed in the first part, with distillation before pruning. There is no notable difference. However, we can note that the model sizes, which were almost identical across tasks, are slightly different this time.

It appears that distillation does not present a major advantage. However, we could consider whether removing layers to complement the student model with new layers is a way of adding adapters. Therefore, models that have been distilled might generalize better and achieve better efficiency on different tasks. It would be interesting to test this in the future and draw parallels with adapters.

5.2.1 SQUAD

Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable.
Results on SQUAD :
I collected the data for this task without pruning.
Firstly, accuracy could be improved. However, we already have a relatively high inference time. And the emission rate is rather average. Comparable to what we obtained with the MNLI spot.

Accuracy	f1 score	inference time (s)	model size	emissions (mJ)
52.7057	65.33	130.06	108893186	3.206

This part is not completely finished. The pruning has yet to be added.

6 Adapters

6.1 State of the art

BERT adapters were introduced by the following paper *Parameter-Efficient Transfer Learning for NLP*[10]
The proposed adapter model adds new modules between layers of a pre-trained network called adapters. This means that parameters are copied over from pre-training and only a few additional task-specific parameters are added for each new task, all without affecting previous ones.
A small number of parameters are introduced in the proposed adapter-based tuning architecture, with the intention of keeping the original network unaffected and the training stable. The approach is to initialize the adapters to a near-identity function so that it can influence the distribution of activations while training.
The adapters project the original feature size to a smaller dimension and then projects them to the original size thereafter, ensuring that the number of parameters stays substantially small as compared to the original model. With the reduction of parameters,

[10]`https://arxiv.org/abs/1902.00751`

there is an obvious **trade-off between performance and parameter efficiency** which is discussed in the experiments below.

	+	−
Fine Tuning	• adapts to specific tasks • high performance	• significant computational resources • over-fitting • the data set for the task must be large
Adaptor	• adding an ”adapt” layer to a pre-trained model • light in terms of computational requirements • retains the basic model: better generalisation on tasks • reduced time	• less efficient

6.2 Adapter on BERT

I am presenting the idea and the model architecture in the following figures :

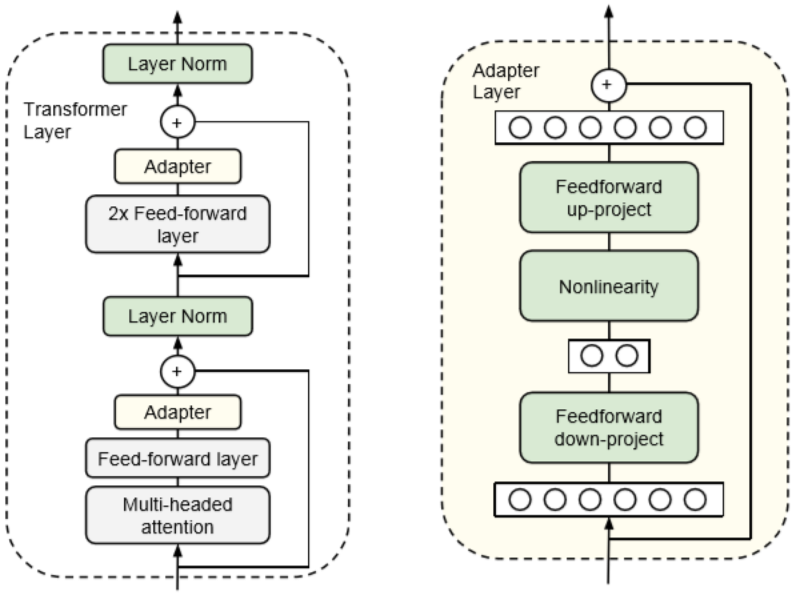
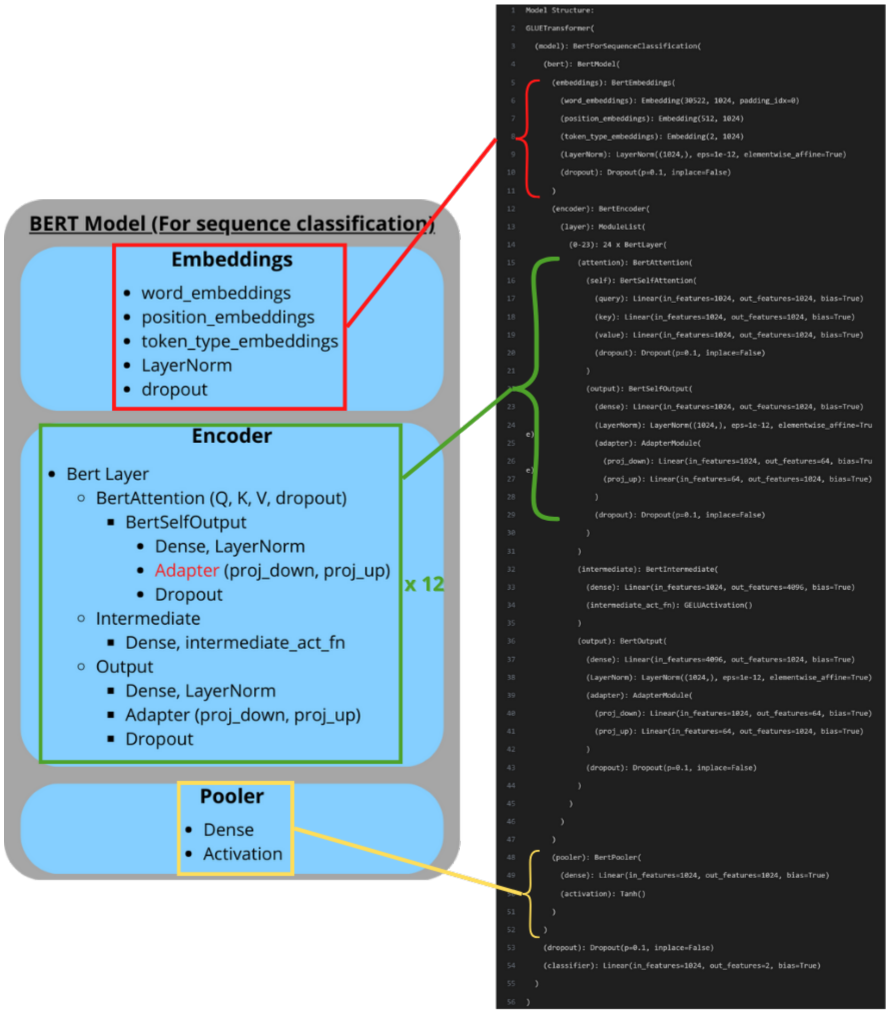


Figure 22: Architecture of the adapter module and its integration with the Transformer. **Left:** We add the adapter module twice to each Transformer layer: after the projection following multiheaded attention and after the two feed-forward layers. **Right:** The adapter consists of a bottleneck which contains few parameters relative to the attention and feed-forward layers in the original model. The adapter also contains a skip-connection. During adapter tuning, the green layers are trained on the downstream data, this includes the adapter, the layer normalization parameters, and the final classification layer (not shown in the figure)



```
Model Structure:
GLUETransformer(
  (model): BertForSequenceClassification(
    (bert): BertModel(
      (embeddings): BertEmbeddings(
        (word_embeddings): Embedding(30522, 1024, padding_idx=0)
        (position_embeddings): Embedding(512, 1024)
        (token_type_embeddings): Embedding(2, 1024)
        (LayerNorm): LayerNorm((1024,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (encoder): BertEncoder(
        (layer): ModuleList(
          (0-23): 24 x BertLayer(
            (attention): BertAttention(
              (self): BertSelfAttention(
                (query): Linear(in_features=1024, out_features=1024, bias=True)
                (key): Linear(in_features=1024, out_features=1024, bias=True)
                (value): Linear(in_features=1024, out_features=1024, bias=True)
                (dropout): Dropout(p=0.1, inplace=False)
              )
              (output): BertSelfOutput(
                (dense): Linear(in_features=1024, out_features=1024, bias=True)
                (LayerNorm): LayerNorm((1024,), eps=1e-12, elementwise_affine=Tru
e)
                (adapter): AdapterModule(
                  (proj_down): Linear(in_features=1024, out_features=64, bias=Tru
e)
                  (proj_up): Linear(in_features=64, out_features=1024, bias=True)
                )
                (dropout): Dropout(p=0.1, inplace=False)
              )
            )
            (intermediate): BertIntermediate(
              (dense): Linear(in_features=1024, out_features=4096, bias=True)
              (intermediate_act_fn): GELUActivation()
            )
            (output): BertOutput(
              (dense): Linear(in_features=4096, out_features=1024, bias=True)
              (LayerNorm): LayerNorm((1024,), eps=1e-12, elementwise_affine=True)
              (adapter): AdapterModule(
                (proj_down): Linear(in_features=1024, out_features=64, bias=True)
                (proj_up): Linear(in_features=64, out_features=1024, bias=True)
              )
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
        )
      )
      (pooler): BertPooler(
        (dense): Linear(in_features=1024, out_features=1024, bias=True)
        (activation): Tanh()
      )
    )
    (dropout): Dropout(p=0.1, inplace=False)
    (classifier): Linear(in_features=1024, out_features=2, bias=True)
  )
)
```

Figure 23: Architecture of our BERT model : with adapter layers

6.3 Results : Adapter and Pruning

6.3.1 Model size

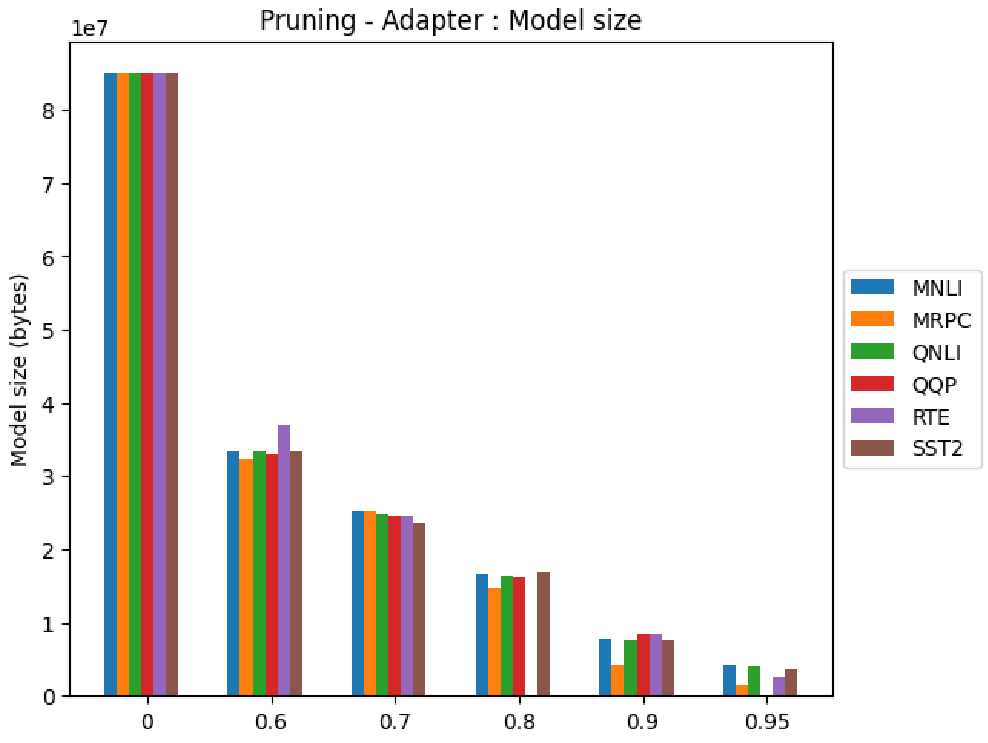


Figure 24: Reduction of the model size

We observe that the model size is quite similar to what we had previously obtained in the section that used fine-tuning. However, upon closer inspection, we notice a slight increase in size. Let's take the QNLI task as an example.

Sparsity	0.6	0.7	0.8	0.9	0.95
Classic Fine tuning	33467323	24739654	16328000	7632995	4102493
Adaptor	33697946	25024805	16370060	8504751	4340502

This increase is certainly due to the addition of additional adapter layers. Specifically, we add (2 adapters (BertSelfOutput) + 2 adapters (Output) $\times$ 12 layers = 48 layers in total. However, this difference remains negligible, and we cannot consider this gap in our analysis because it is too small to be significant. Therefore, we will consider that the models derived from fine-tuning and those derived from the adapter method are identical.

`Accuracy for the sparsity ratio of 0.8 :`

Task	SST2	QNLI	QQP	MNLI	MRPC	RTE
Accuracy	0.857	0.852	0.823	0.803	0.71	0.584

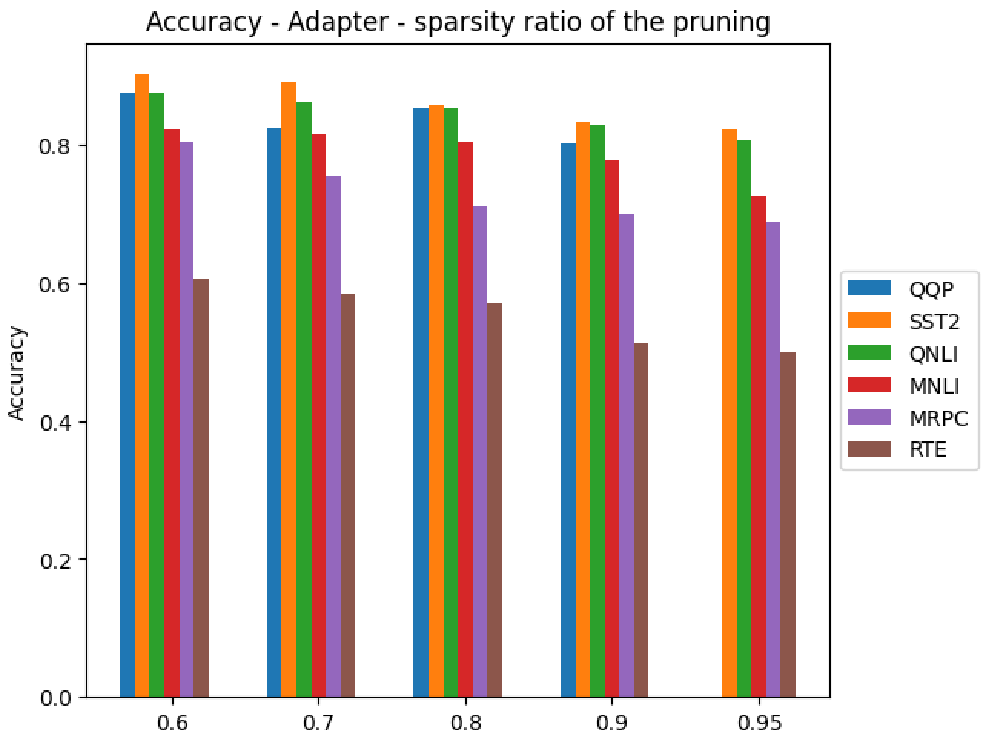


Figure 25: Accuracy and sparsity

6.3.2 Speedup

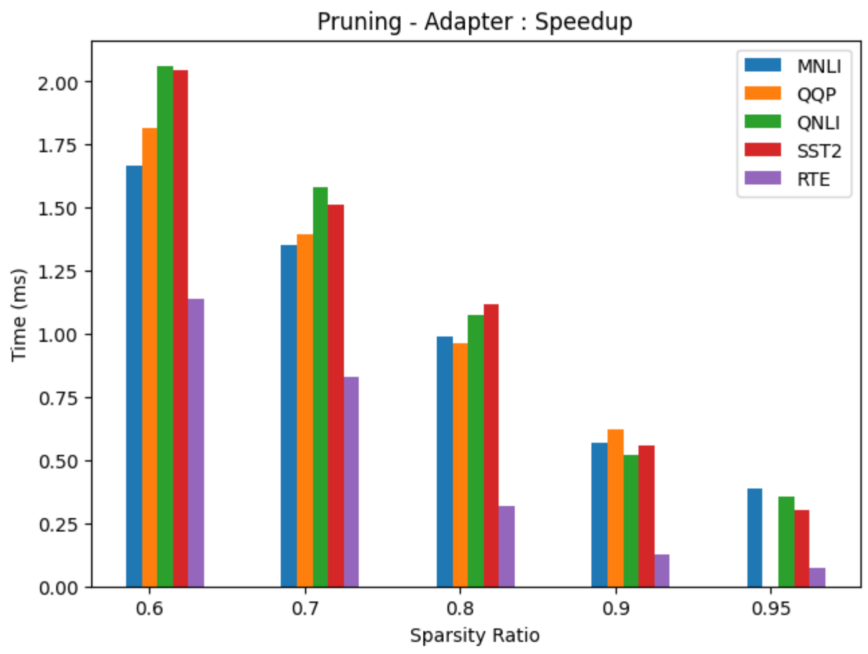


Figure 26: Inference time of each task depending on the sparsity

In this figure, the MRPC task is not depicted because, as previously mentioned, it is significantly longer than the others. I have included it in the graph below. Generally, we observe a reduction in time by almost 2 times. This aligns well with our expectations.

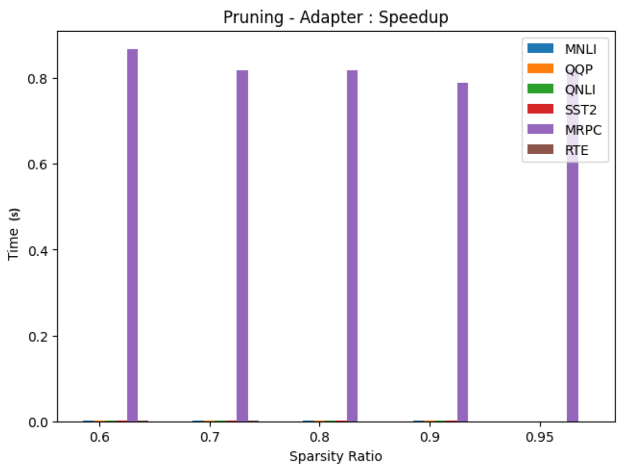


Figure 27: Inference time of each task depending on the sparsity

We can also visualise the training time.

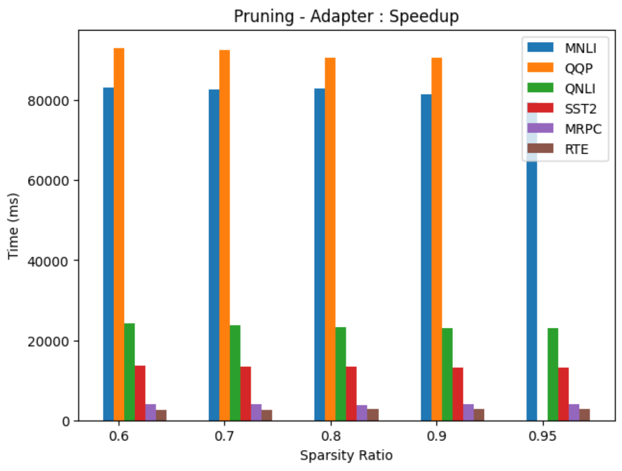


Figure 28: Training time of each task depending on the sparsity

6.3.3 Emissions

Comparison between the emissions and the model size according to the sparsity target.

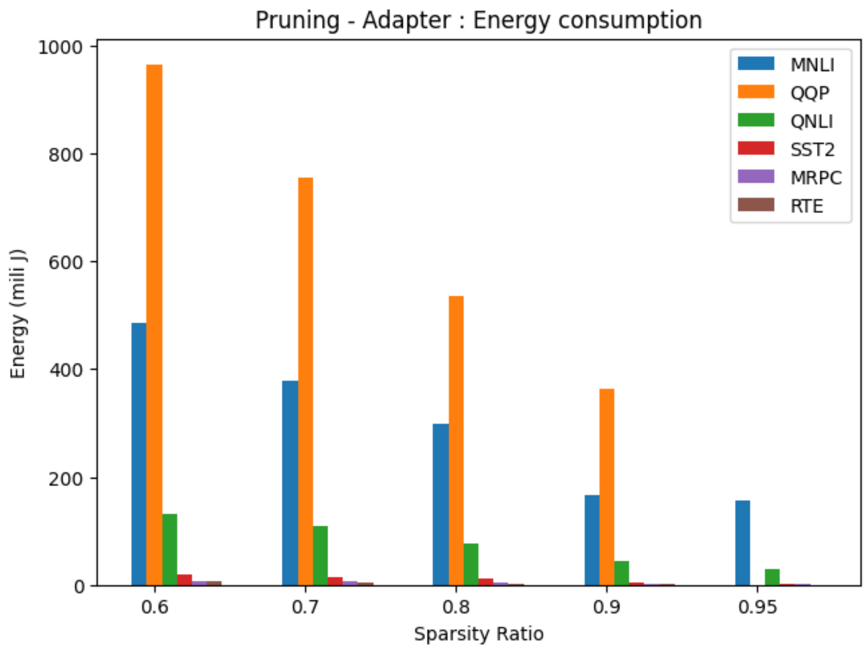


Figure 29: Energy and model size for each sparsity

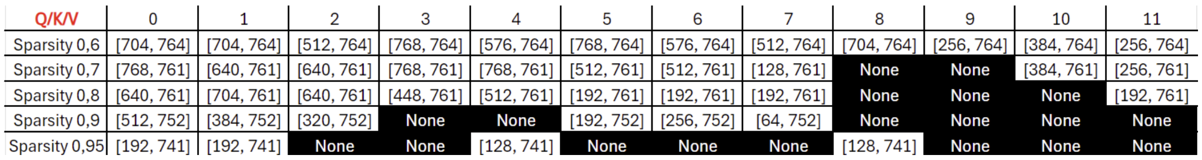
There is a reduction in energy costs with the addition of adapters, unlike the traditional use of fine tuning. This was predictable because we had also observed a decrease in time. As we have seen, there is a strong correlation between time and emissions.

Thus, adapters offer several advantages:

- Reduction in energy emissions
- Decrease in inference time
- Accuracy remains satisfactory, as there is no difference compared to what was achieved with fine tuning

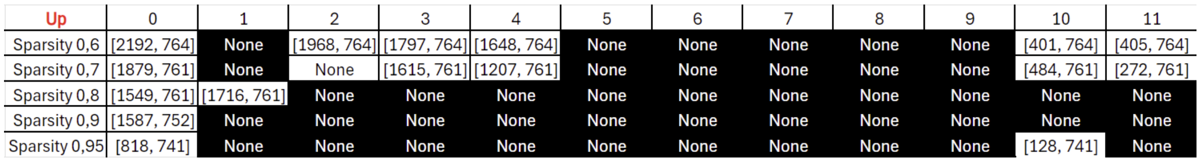
6.3.4 Layers

Similary to what I have done previously, let's look at the following layers :Query, Key, Value, Up, Down and Output layer. $(size(Query) = size(Key) = size(Value) = transpose(Output)$ and $size(Down) = transpose(Up))$
Here, we focus on the QNLI task as in the previous section. I record the dimension of each layer for every sparsity level. "None" indicates that the layer has been completely pruned and thus removed.



Q/K/V	0	1	2	3	4	5	6	7	8	9	10	11
Sparsity 0,6	[704, 764]	[704, 764]	[512, 764]	[768, 764]	[576, 764]	[768, 764]	[576, 764]	[512, 764]	[704, 764]	[256, 764]	[384, 764]	[256, 764]
Sparsity 0,7	[768, 761]	[640, 761]	[640, 761]	[768, 761]	[768, 761]	[512, 761]	[512, 761]	[128, 761]	None	None	[384, 761]	[256, 761]
Sparsity 0,8	[640, 761]	[704, 761]	[640, 761]	[448, 761]	[512, 761]	[192, 761]	[192, 761]	[192, 761]	None	None	None	[192, 761]
Sparsity 0,9	[512, 752]	[384, 752]	[320, 752]	None	None	[192, 752]	[256, 752]	[64, 752]	None	None	None	None
Sparsity 0,95	[192, 741]	[192, 741]	None	None	[128, 741]	None	None	None	[128, 741]	None	None	None

Figure 30: Layers size of Query, Key and Value



Up	0	1	2	3	4	5	6	7	8	9	10	11
Sparsity 0,6	[2192, 764]	None	[1968, 764]	[1797, 764]	[1648, 764]	None	None	None	None	None	[401, 764]	[405, 764]
Sparsity 0,7	[1879, 761]	None	None	[1615, 761]	[1207, 761]	None	None	None	None	None	[484, 761]	[272, 761]
Sparsity 0,8	[1549, 761]	[1716, 761]	None	None	None	None	None	None	None	None	None	None
Sparsity 0,9	[1587, 752]	None	None	None	None	None	None	None	None	None	None	None
Sparsity 0,95	[818, 741]	None	None	None	None	None	None	None	None	None	[128, 741]	None

Figure 31: Layers size of Up layer

In these two tables, which we compare with those seen previously, we notice several things. First of all, more layers have been pruned. The first layers are always retained. Additionally, this time the last 3 layers are not removed even at a sparsity of 0.6.
We could explain the increase in pruned layers by noting that, by adding the adapters, we have concentrated the essential information within the adapters. This makes the "other" layers less necessary, as they no longer carry the primary information. This also helps us understand why the model size has not significantly changed between the model trained with fine-tuning and the model trained with adapters.
We also observe that the up, down, and output layers are pruned more compared to the query, key, and value layers.
Let us recall that the adapters are added in BertSelfOutput, which also contains Q and V, and also in Output, which contains Up and Down. Could there be a connection? It seems as though the up and down layers carry less information.

7 Conclusion

In this work, I focused on several issues, some of which can still be further explored. What are the optimisation methods used in NLP ? How can we improve pruning ? What are the pruning methods ? How can we reduce the energy cost of training ? And more generally, what about models and learning processes ?

In this work, I focused solely on the BERT model (bert-base-uncased). The aim was to explore various possibilities for improving pruning and optimization. The choice of parameters depends on what we aim to prioritize: speedup, model size, accuracy, or reducing energy cost.
The initial objective was to perform structured pruning. Then, we aimed to add elements to enhance pruning. With structured pruning, we achieved the desired results: we can accelerate acquisition time, reduce energy costs, and decrease model size without losing accuracy.
In the second phase, instead of classic fine-tuning as in the first part, we used adapters. By adding these layers to our model, we can improve performance in terms of task generalization, speedup, and reduce energy costs even more.
In this work, I also introduced distillation before structured pruning. However, distillation does not appear to be very beneficial according to the results obtained. Adding distillation is a double-edged sword: it improves performance slightly, but training this model requires more iterations and therefore more time.
Adapters are quite interesting in our case and seem to provide significant gains during pruning. They allow us to maintain the model's skeleton.

To go further: I assumed that adapters enable task generalization. When we retrain the model on another task, we do not modify the model's skeleton but only the adapter layers. This way, we can train our model on multiple tasks consecutively without affecting the accuracy of previous tasks. There are other adapter methods, such as LORA (Low-Rank Adaptation), which are claimed to be more efficient. In this work, I focused on the basic adapter model.
Moreover, I mainly focused on GLUE tasks because they are easy to implement and provide a preliminary understanding of our model's effectiveness. I could have further tested other tasks such as SWAG, WikiText, WMT, Natural Questions, RACE (Reading Comprehension from Examinations).