TÉCNICO LISBOA

Best Way to Squeeze: A Comparison of Model Compression Techniques in Natural Language Processing

João Carlos Lopes Antunes

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor(s): Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Prof. Miguel Filipe Leitão Pardal

Examination Committee

Chairperson: Prof. João António Madeiras Pereira
Supervisor: Prof. Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Member of the Committee: Prof. David Rogério Póvoa de Matos

October 2021

Dedicated to Beatriz.

Acknowledgments

This project would not have been possible without the constant support from my supervisors, Prof. Luísa Coheur and Prof. Miguel Pardal, whom I would like to deeply thank. Their guidance and continuous encouragement throughout the development of this project in its entirety was fundamental, and their constant feedback over my numerous revisions pushed the quality of this project to the best it could be.

I would also like to thank Ricardo Rei from the Unbabel research team for providing his own researched models and the coding insight behind them, and whose opinions helped shape and improve the experimentation process of this project.

And last but not least, many thanks to all of my friends and family members, which helped me endure this long and arduous task by offering support and stress relief.

Resumo

Os desenvolvimentos mais recentes no processamento de linguagem natural resultam em cada vez mais modelos treinados intensivamente com grandes quantidades de recursos computacionais, com o objetivo de se sobreporem e ultrapassarem os modelos do estado-da-arte em termos de resultados obtidos. No entanto, o treino que estes modelos de alto desempenho requerem pode durar várias horas, dias, ou mais tempo ainda. Para além disso, a complexidade dos modelos e dos recursos necessários para os treinar e executar proíbe a sua utilização em dispositivos mais limitados em termos de memória, poder de processamento e latência na capacidade de resposta.

Neste trabalho, focámos a nossa atenção no esforço necessário para treinar estes modelos de alto desempenho, através da análise destes mesmos modelos e da recolha de informação acerca dos recursos computacionais necessários, dos conjuntos de dados usados, e da complexidade dos modelos. Aplicámos técnicas de compressão a modelos do estado-da-arte para criar versões compactas desses mesmos modelos, a fim de comparar e avaliar a eventual perda de desempenho do modelo face à simplicidade do modelo e à redução de recursos computacionais e gastos energéticos.

Palavras-chave: aprendizagem automática, redes neuronais, processamento de linguagem natural, compressão de modelos, avaliação de modelos, pegada ecológica

Abstract

Current research in natural language processing shows a growing number of models extensively trained with large computational budgets, pursuing the goal of outperforming other state-of-the-art models in test-set performance scores. However, with such computationally demanding requirements, training these models often requires several hours, days, or worse. Furthermore, the sheer complexity and resources required to evaluate such models prevents them from being deployed in devices with strict resource and response latency limitations.

In this thesis, we focus our attention on the effort required to train and evaluate such high performing models. We analyze several of the latest proposed models, gather information about their computational budgets, datasets used and model complexity, and apply state-of-the-art model compression techniques to create compact versions of those models. We then evaluate whether the trade-off between model performance and budget is worthwhile, in terms of evaluation efficiency, model simplicity and environmental footprint.

Keywords: machine learning, neural networks, natural language processing, model compression, model evaluation, environmental footprint

Contents

List of Tables

List of Figures

Acronyms

BERT Bidirectional Encoder Representations from Transformers.

BLEU Bilingual Evaluation Understudy.

DQ Dynamic Quantization.

EEA European Environment Agency.

GLUE General Language Understanding Evaluation.

GPT Generative Pre-trained Transformer.

IMP Iterative Magnitude Pruning.

LM Language Modeling.

NLG Natural Language Generation.

NLP Natural Language Processing.

PKD Patient Knowledge Distillation.

PUE Power Usage Effectiveness.

QAT Quantization-Aware Training.

SQuAD Stanford Question Answering Dataset.

TER Translation Error Rate.

U.S. EPA United States Environmental Protection Agency.

Chapter 1

Introduction

The field of machine learning has brought a revolution in performing human-like tasks through computing, especially with the research and development in neural network technology. With the advent of better technology and computational capability, the resources available to train and execute such neural networks have been steadily improving over the past decade; as such, the current trend towards bigger and better models suited for Natural Language Processing (NLP) tasks is understandable to see. This is especially true for recent models which are overwhelmingly based on the Transformer [1], a neural network architecture that has revolutionized NLP with its well-performing generalization to several different NLP tasks. Despite all of these incredible advances, this architecture favors bulky models containing a big number of parameters – model weights and coefficients that change and learn with the training data – which require large amounts of computational resources to both train and run. Looking at the two previous years, we have seen several large-scale transformer-based models, ordered here by their number of parameters[1]:

- **BERT** [2] – 340 million parameters[2];
- **GPT-2** [3] – 1.5 billion parameters[3];
- **Megatron-LM** [4] – combined total of 8.3 billion parameters;
- **Turing-NLG** [5] – 17 billion parameters;
- The latest model from the *GPT-n* models, **GPT-3** [6] – a staggering 175 billion parameters.

Models with this enormous scale have an extremely large computational overhead and memory consumption, not to mention the energy cost required to perform any sort of task. Nonetheless, this trend will most likely continue for the foreseeable future as the results speak for themselves: all of the aforementioned models set new state-of-the-art results; as mentioned by Microsoft in their *Turing-NLG* blog post [5], *"larger natural language models lead to better results"*.

[1] Following the U.S. convention of a billion (10^9).

[2] Parameters in the largest *BERT* model, *BERT*$_{LARGE}$. The default *BERT*$_{BASE}$ uses 110 million parameters instead.

[3] Parameters used in the largest *GPT-2* model, *GPT-2 XL*. The commonly used *GPT-2*$_{Small}$ has a more modest 124 million parameters.

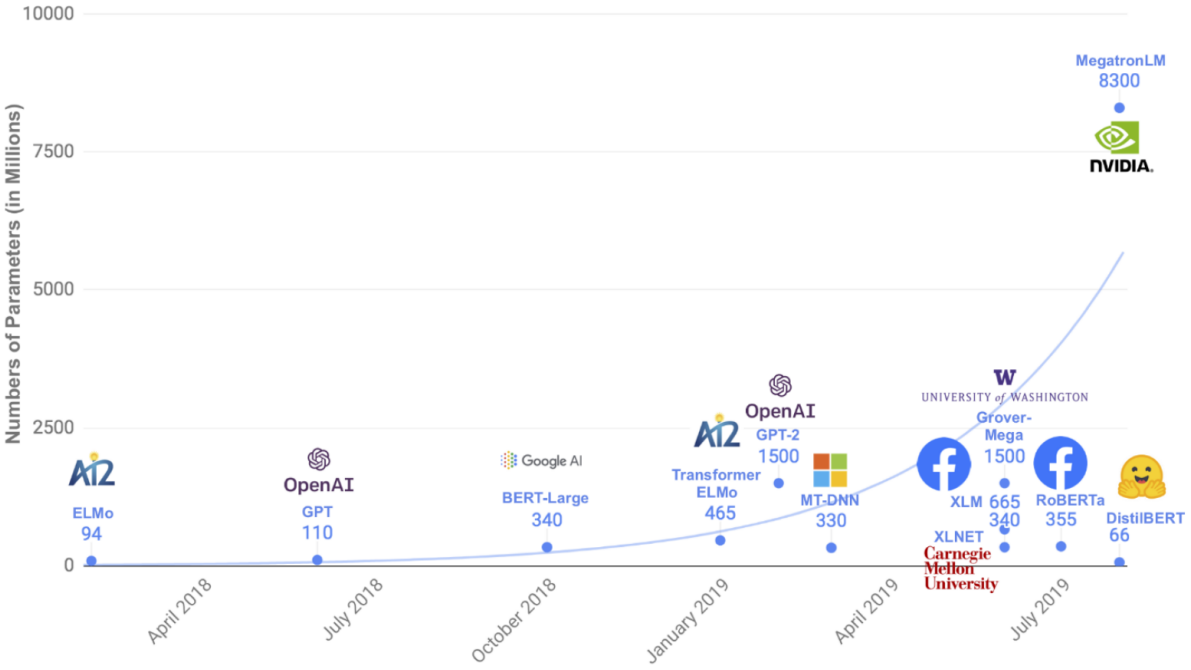


Figure 1.1: Number of parameters of some of the larger NLP models released between 2018 and 2019. Source: [7]

1.1 Motivation

The issue with such high requirements for high-performing language models becomes more apparent once the scale of the hardware is taken in consideration; fitting a model with a billion parameters in a single advanced data-center GPU, such as the *NVIDIA Tesla V100* with 32GB of memory, is impossible. Several approaches tackle this problem by using *model parallelism* to partition very large models over several GPUs, such as using a framework like *Mesh-Tensorflow* [8]; this same parallelism technique was used to train the 8.3 billion parameters of the *Megatron-LM* model over 512 GPUs [4]. However, even with parallelism, the *memory consumption* remains a major problem for less advanced hardware used outside of data-centers, especially if we consider devices with low-end hardware (such as smartphones, that have to be portable, have limited cost, and rely on battery for power).

Another concern with executing such large models is the *time* they take to infer a result – the latency of the model. Similarly to the training process, model latency is heavily dependent on the environment where the model is deployed; using low-performing hardware results in slower inference times in large models. Basic benchmarking done on a *GPT-2* model deployed to AWS [9] showed that the usage of a GPU provides substantially lower latency than the usage of a CPU for model inference; this can impact the usefulness of the model in performing tasks that require very low inference time, such as auto-completing word typing. The hardware requirements become increasingly worrisome if the model is not deployed on the cloud but instead on devices with hardware constraints, such as the integrated GPUs often used in smartphones; this effectively leads to an undesirable trade-off between model accuracy and response latency.

Benchmark	Error Rate	Polynomial		Exponential	
		Environmental Cost (CO_2)	Economic Cost ($)	Environmental Cost (CO_2)	Economic Cost ($)
SQuAD 1.1	Today: 4,621%	10^4	10^5	10^5	10^5
	Target 1: 2%	10^7	10^7	10^{15}	10^{15}
	Target 2: 1%	10^{10}	10^{10}	10^{32}	10^{32}
CoLLN 2003	Today: 6,5%	10^5	10^5	10^5	10^5
	Target 1: 2%	10^{35}	10^{35}	10^{73}	10^{74}
	Target 2: 1%	10^{53}	10^{53}	10^{173}	10^{173}
WMT 2014 (EN-FR)	Today: 54,4%	10^4	10^4	10^4	10^4
	Target 1: 30%	10^{15}	10^{15}	10^{22}	10^{22}
	Target 2: 10%	10^{35}	10^{35}	10^{99}	10^{100}

Table 1.1: Implications of achieving performance benchmarks on carbon emissions (lbs) and economic costs ($USD) from deep learning in NLP (namely, *question answering*, *named entity recognition* and *machine translation*) based on projections from polynomial and exponential models. The carbon emissions and economic costs of computing power usage are calculated using the conversions from [10]. Adapted from source: [11]

When expanding the scope of language model resources onto the real world, there are some concerns that cannot be ignored but are often overlooked in an academic perspective: the *monetary cost* and the *environmental footprint*. Training a large, state-of-the-art model is not cheap. A recent review [12] of the cost of training differently sized *BERT* models estimates that training a model with 1.5 billion parameters costs around $80k[4] for a single training run; if considering proper hyperparameter tuning and 10 training runs, the cost escalates to around $1.6m. These alarming figures understandably act as a paywall for developing new models, as not many research labs can afford costs of this magnitude; this leads to an increase in popularity in fine-tuning large models to specific tasks, while letting "the big players" actually train the models and make them available to the public. However, we are reaching a point where not even those who can make such high monetary investments are willing to do it: when reviewing datasets for testing *GPT-3*, the authors of the model found a mistake made when implementing the system, but decided against fixing it since *"due to the cost of training it was not feasible to retrain the model"* [6]. There is also a considerable environmental cost in heavily training models: a recent study [10] reported that a fully trained *BERT* model, taking about 79 hours to train using cloud computing GPUs, emits roughly as much CO_2 as a trans-American flight from New York to San Francisco. Not only that, but future models may bring much more drastic effects to the environment; using projections based on the computational costs required to hit current state-of-the-art benchmark results (shown in Table 1.1), we can expect both the environmental and economical cost of training a deep learning model for NLP usage to be higher in the order of a few magnitudes at best, or hundreds of magnitudes at worst, not to mention the necessary computational requirements to achieve such high energy expenditure.

[4]Costs denoted in U.S. dollars

1.2 Objectives

As an effort to tackle the issues previously explained, the objectives of this work lie in the study of model compression techniques applied on language models and the results thereof.

We propose a *comparison of three different compression techniques* when used on language models, with each of them following a separate compression methodology: *knowledge distillation*, *weight pruning* and *model quantization*. To do so, we apply each of these compression techniques on three different language models, fine-tuned to perform very common NLP tasks – *sentiment analysis*, *named entity recognition* and *dialog-driven sentence generation*[5] – and compare the resulting compressed models, determining whether the trade-off between performance and resource usage is worth it. We also propose combining two of the compression techniques, quantization and pruning, to compare the resulting compressed language models against the separately compressed models for both techniques.

To answer whether large models can be compressed to obtain better performance without severely degrading model quality, we propose a well-detailed evaluation of every model: we display the expected validation performance for all evaluation metrics, the final size of the model, average training and inference time, computing infrastructure used, model hyperparameters and dataset splits, as well as a link to model implementation code. We verify the inversely proportional correlation between execution latency and the size and complexity of the compressed model. Furthermore, we propose a brief comparison between uncompressed and compressed models when running in low-end hardware by testing the performance of quantized models in a *Raspberry Pi*.

Additionally, we compare the training process in terms of training time and power usage, as well as provide an estimation of CO_2 emissions for the same model to be trained in a data center, to understand whether the reduction in model size and complexity – the *performance squeeze* mentioned in the title of this work – also translates to a decrease in the computational budget required to further train and fine-tune the model, as well as a decrease in the environmental footprint of the overall training process.

1.3 Main Contributions

The main contributions of this work are the following:

- Provide a fair comparison between common model compression techniques across the same base models, that can be applied to state-of-the-art language models. This is achieved by only utilizing compression techniques that do not require a fundamental change to the architecture of the model in order to work, as well as guaranteeing that the models are trained and tested on the same hardware.

- Test compressed models in an environment with limited resources. Besides evaluating models in a standard GPU-rich environment for large-scale models, we made use of compression techniques that create CPU-oriented models and tested those same models in a low-end device.

[5]The work reported in this dissertation relates with the research target of project MAIA (CMU-PT MAIA Ref. 045909).

- Make every experiment available for further study and usage. All of our code repositories are open-source and made available on the GitHub platform (listed in Section 4.1).

1.4 Thesis Outline

In Chapter 2, we describe in detail the main concepts of model compression, as well as some state-of-the-art techniques applicable to language models; in Chapter 3, we present some of the active and more recent work being done in model compression and provide a more in-depth overview on these studies, their results and possible shortcomings. We detail our experiments in using model compression techniques on recent NLP models, as well as our approach to evaluating said models in Chapter 4, and, in Chapter 5, we compare our results and draw out a conclusion on the subject of model quality versus computational budget. Finally, we outline the main findings of this study and mention the advantages of applying model compression techniques to current models in Chapter 6.

Chapter 2

Background

In this section, we explain the concepts behind the most common model compression techniques used today, as well as their advantages and drawbacks.

Current high performing models are usually slow in execution and have large space and memory requirements. As such, *model compression techniques* [13] aim to lower the resources necessary for the model to perform, thus reducing the budget required to train and execute the model while retaining as much accuracy as possible compared to the original, uncompressed model. Several of these techniques have been proposed and studied over the past years, with some of them becoming more common recently: *knowledge distillation*, *pruning* and *quantization*.

2.1 Knowledge Distillation

Knowledge distillation [14, 15], also known as teacher-student learning, refers to the training of a small, compact model – *the student* – to approximate the knowledge learned by a highly-parameterized complex model, trained over massive amounts of unlabeled data – *the teacher*. This larger model has higher knowledge capacity and can provide high performance but is also computationally expensive to evaluate. By training a fast and compact model on a separate dataset – called a *transfer set* – while regulating the training using the soft outputs provided by the larger model's output layer, the student model starts learning the so-called "dark knowledge" of the teacher model: it learns to mimic the output of the larger model, therefore approaching the function learned by the teacher model without having to be trained on the massive dataset that made the model end up with said function. Hopefully, the student model ends up performing only slightly worse than the teacher model, while substantially lowering the computational budget required to execute it. Knowledge distillation is also independent from the architecture of the model, since it depends solely on the output provided by the teacher model, making it a very versatile model compression technique to apply.

Some work has also been done in further knowledge extraction from the teacher model. *Patient knowledge distillation* [16] is an approach to teacher-student learning where the student model extracts information from intermediate layers of the teacher model, and not exclusively from the output layer. This

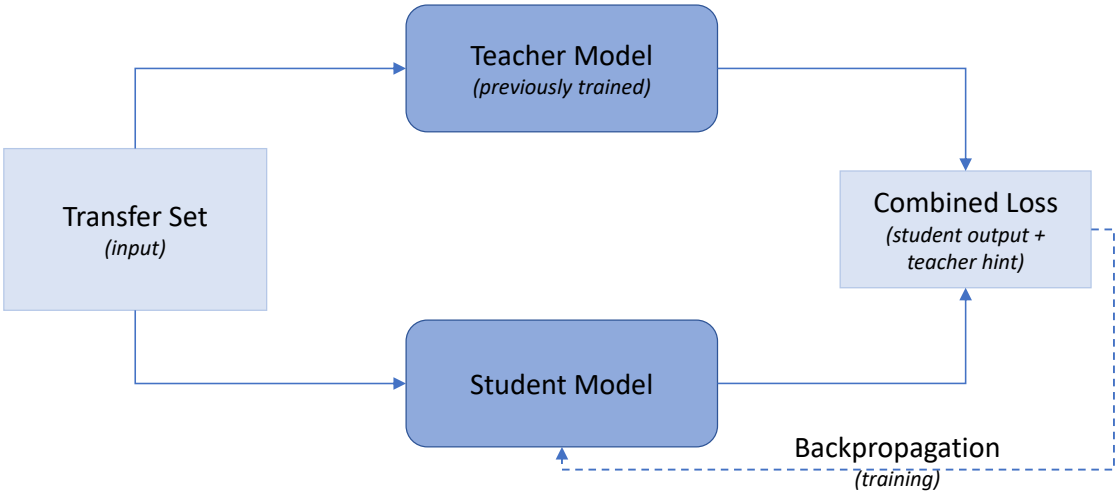


Figure 2.1: An overview of teacher-student learning.

patient learning approach can follow one of two approaches: either the student focuses on learning from the last k layers, or it learns from every other k layers. This multi-layer distillation process shows an improvement in accuracy regarding regular knowledge distillation approaches, while still compressing the original model as intended.

Although knowledge distillation as a model compression technique originally focused on the training of a fast and compact model from scratch, research has been done in applying knowledge distillation to pre-trained models instead [17]. This form of *pre-trained distillation* has the compact student model pre-trained on unlabeled language model data, turning the model into a well-read student. The student can now take full advantage of the teacher's soft label outputs while training over the transfer dataset, since pre-training mitigates the initial error present in the otherwise randomly initialized distillation process. A pre-trained distilled model can then be subjected to fine-tuning, to make the model more robust for the task at hand.

2.2 Pruning

Network *pruning* refers to the removal of redundant and non-informative connections within the network of the model, thus achieving a reduction in model size and potential improvements in execution time and energy efficiency, while only slightly degrading the quality of the model [18]. Despite the improvements that can be obtained, such pruned models often end up with sparse connection matrices, which have an additional storage overhead compared to regular matrices and require purpose-built hardware capable of efficiently loading and performing operations; these details must be taken into consideration to consider the overall trade-off.

Very early works on the pruning of neural networks performed pruning by first computing an approximation of the loss function of the model with respect to its parameters. This allows for an iterative checking of increases in the loss function of the model when setting a given weight to zero, thus calcu-

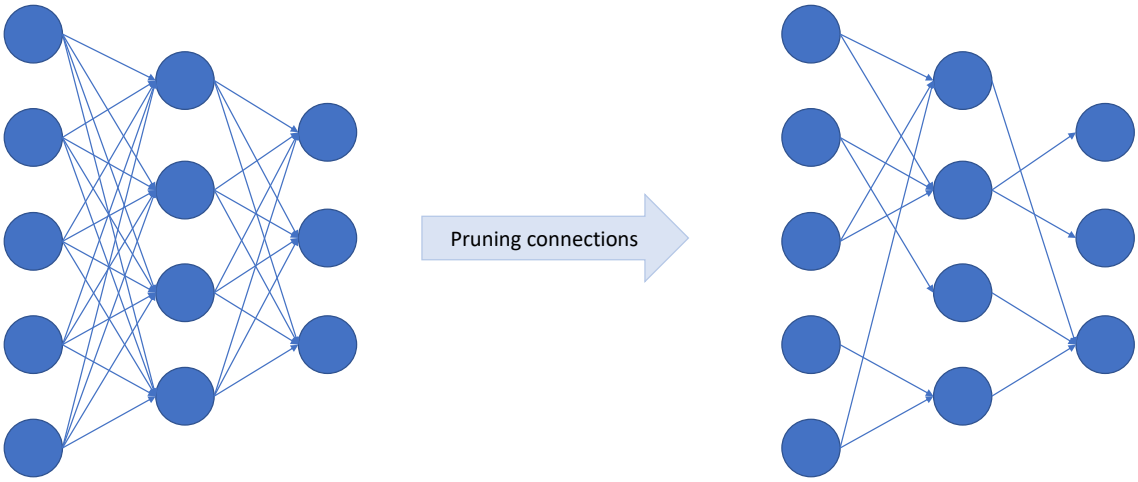


Figure 2.2: Representation of pruning applied to a network (weight-based pruning).

lating the importance of that weight to the network - its *saliency*. A parameter with small saliency will have a minimal effect on the training error if it is removed. Two of the most popular techniques, *Optical Brain Damage* (OBD) [19] and *Optimal Brain Surgeon* (OBS) [20], compute a complex approximation of the loss function of the model and use it to determine the parameters with low saliency, which are then pruned from the network; the model is then retrained. Both techniques show significant improvements to the inference time and accuracy of the neural network, but applying these techniques is unfavorable due to the high computational complexity of the approximation.

More recent approaches [18, 21] use *magnitude-based* weight pruning. This technique consists in picking a percentage of weights to be pruned, and removing that same percentage of weights with values closest to zero [22]. Despite creating pruned models with worse results in accuracy than those obtained from both OBD and OBS, magnitude-based approaches are often preferred due to being computationally efficient and scaling better to large models and datasets, which are particularly common in recent years.

2.3 Quantization

A logical step in compressing a model would be to trim down the model parameters themselves. Network *quantization* [23] refers to the compression of the weights present in the inner layers of the model, such that the weight values can be represented using a smaller amount of bits. The parameter matrices end up occupying less memory and any required arithmetic operations become faster as a result. Previous works have shown that both network pruning and quantization are effective not only in lowering the complexity of the model, but also as a way to address over-fitting. Furthermore, the two techniques are compatible with each other and can be used simultaneously for further model reduction.

A simple but extreme way of performing quantization forces every weight value to be represented by a single bit, effectively turning them into binary weights – hence called *binarization*. This can be done by using the sign of the weight value: any positive values become 1, and any negative values become 0.

Binarization is applied to neural networks from scratch, so they can directly learn binary weights during the model training. Although it provides a meaningful compression to the data of the model, binarization tends to drastically reduce the accuracy of the model; this reduction is significantly noticeable when dealing with large neural networks.

Another simple method of quantization is a method of weight sharing, called *scalar quantization* [24]: by clustering the weight values using *k*-means clustering, each weight can be represented by one of the *k* cluster centers, and gets assigned an index for that specific cluster. The calculated cluster centers can then be stored separately in a *codebook*, whose values can be looked up directly whenever necessary. Using this approach, parameters can be compressed into the $log_2(k)$ bits necessary to represent each cluster index, while the uncompressed cluster values are stored in the codebook. Considering the fast value lookup on the codebook and the negligible size of the codebook, even for a relatively high number of *k* clusters, this approach gives a good performance for compressing parameters, especially in models with large weight matrices.

Generally, these quantization techniques are applied post-training; this allows for immediate compression of the model in its fully trained state, lowering the size of the model. However, for most of these techniques, the model itself still computes using floating-point arithmetic operations, so no improvements to the model latency are achieved. Furthermore, the quantized model does not take in consideration the error obtained from losing precision in the weights of each layer, resulting in slightly worse accuracy results. To avoid a significant drop in model quality derived from the loss in accuracy and latency, we can employ quantization during the training process – *Quantization-Aware Training* (QAT). We explain quantization-aware training in further detail in Section 3.2.1. Models trained this way have smaller accuracy losses over the base model compared to post-training quantization.

2.4 Summary

Model compression techniques study different ways of reducing the size occupied by a model, as well as the computational resources necessary to train and execute the model, while maintaining as much performance of the original model as possible.

Knowledge distillation focuses on training a small and compact model by supervising its learning with a large and well-trained model. The smaller model inherits the knowledge of the larger model by evaluating against the outputs of the larger model during the training process, thus resulting in a faster and compact model with the quality and performance of the initial larger model.

Pruning tackles the outright removal of some of the weights present in the model layers, which reduces the number of trainable parameters in the model, theoretically allowing for faster inference times and reduced model size. However, in practice, the size reduction is limited by the method of storing model weights, and efficient sparse matrix storage is only available for very highly sparse matrices [25].

Quantization studies the size compression on the scale of weight values, by either clustering similar parameters and sharing them across the weight matrices or reducing the bit representation of those values (for example, from floating-point to integer). These processes tend to drastically reduce model

size at the cost of a degradation in model quality, but have the benefit of being able to be applied post-training. Several different quantization techniques have been developed to be applied during training instead, which helps reduce the loss in model performance.

Chapter 3

Related Work

In this section we take an in-depth look at other works regarding model compression techniques used in NLP, as well as improved ways of reporting results when evaluating models. We also examine the environmental impact of training recent models.

3.1 Better Model Evaluation and Result Reporting

An important but often overlooked part of comparing the performance of language models is the way the experiment details and consequent results are reported. Papers often only report their best accuracy values, but omit important details such as the time spent training the model, or any finely tuned hyperparameter values. Not only that, but displaying the best accuracy values does not confidently portray the performance of a model since accuracy values can often vary depending on several testing factors, such as random weight initialization. With this in mind, a recent novel technique for comparing performance between different language models has been presented: *expected validation performance* [26] as a function of computational budget. For a given model, its performance is calculated via the expected accuracy value obtained from validation data, given a resource budget to train and evaluate n models. This expected value takes in consideration every accuracy value obtained during model development and testing, instead of simply reflecting the best observed value after n evaluations. Not only is this expected value a more accurate reflection of the overall performance of the model, but the measuring technique also requires no extra calculations besides the ones obtained during the model training and fine-tuning; training a model is an iterative procedure, and several results will be obtained throughout model tweaking and testing.

For a given model, trained over a set of n hyperparameter values, we can calculate the expected validation performance of a model as follows:

$$\mathbb{E}[V_n^* \mid n] = \sum_v v \cdot P(V_n^* = v \mid n) \tag{3.1}$$

where V_n^* is the maximum value obtained during model evaluation for the given set of n hyperparameter

values, and $P(V_n^* = v \mid n)$ is the probability mass function for the maximum value. It is worth noting that the computational budget can be measured by other metrics, such as training duration; in this case, V_n^* would denote the maximum value obtained for the given training duration. Another notable detail is that, for singular sets of values (such as comparing a single hyperparameter value, or using the training duration) we get $n = 1$, and the expected value simply becomes the mean of all values.

By displaying the performance of the model as a function of computational budget, we can estimate the resources required to attain a certain expected performance value. This can be useful in scenarios where the budget provided for training a model is limited, or if there is a minimum accuracy value required: the estimated budget for a given performance value can be calculated, and unnecessary budget expenses are avoided by only spending resources below the estimated value.

Considering the advantages obtained from this evaluation approach, we will be using this technique to display our results. Every model trained and evaluated under our working environment will have their results detailed as a graph of the expected validation performance as a function of the training duration.

3.2 Compressing BERT

For the past couple of years, NLP research has thoroughly taken advantage of model development based on large pre-trained models, refining and fine-tuning them into performing different tasks; one of the most widely used models as the starting point is none other than *BERT*. As previously stated, despite being a very versatile and effective model, *BERT* still has a large memory footprint and requires heavy computing during inference; therefore, model compression has been an especially important field of study for any *BERT*-based models.

3.2.1 Q8BERT

One of the compression techniques we previously mentioned tackles both the memory and the latency issue, by reducing the representation of weight values to a smaller quantity of bits while also enjoying faster computation on those smaller bit values: quantization. Based on this, *Q8BERT* [27] was created: a quantized version of *BERT* that achieves a $4\times$ smaller memory footprint while only losing less than 1% accuracy relative to the original model. This was achieved by quantizing all weights within the Embedding and Fully Connected layers – which contain over 99% of the weights present in *BERT* – to 8-bit values.

To quantize a given weight x, the following symmetric linear quantization function was used:

$$\begin{aligned} Quantize(x \mid S^x, M) &= Clamp(\lfloor x \times S^x \rceil, -M, M), \\ Clamp(x, a, b) &= min(max(x, a), b) \end{aligned} \tag{3.2}$$

where S^x is the quantization scaling factor for the weight x, and M is the highest quantized value possible

Dataset	Metric	BERT baseline accuracy (STD)	QAT BERT 8-bit (STD)	DQ BERT 8-bit (STD)
CoLA	Matthew's corr.	**58.48** (1.54)	**58.48** (1.32)	56.74 (0.61)
MRPC	F1	**90** (0.23)	89.56 (0.18)	87.88 (2.03)
MRPC-Large	F1	90.86 (0.55)	**90.9** (0.29)	88.18 (2.19)
QNLI	Accuracy	90.3 (0.44)	**90.62** (0.29)	89.34 (0.61)
QNLI-Large	Accuracy	91.66 (0.15)	**91.74** (0.36)	88.38 (2.22)
QQP	F1	87.84 (0.19)	**87.96** (0.35)	84.98 (0.97)
RTE	Accuracy	**69.7** (1.5)	68.78 (3.52)	63.32 (4.58)
SST-2	Accuracy	**92.36** (0.59)	92.24 (0.27)	91.04 (0.43)
STS-B	Pearson corr.	**89.62** (0.31)	89.04 (0.17)	87.66 (0.41)
STS-B-Large	Pearson corr.	**90.34** (0.21)	90.12 (0.13)	83.04 (5.71)
SQuADv1.1	F1	**88.46** (0.15)	87.74 (0.15)	80.02 (2.38)

Table 3.1: *GLUE* tasks and *SQuAD* datasets, metrics and results for the baseline pre-trained *BERT* model and two quantized versions (quantized to 8-bit weights): Quantization-Aware Training (QAT) and Dynamic Quantization (DQ). Tasks suffixed with *Large* were performed with *BERT*$_{LARGE}$; all others were performed with *BERT*$_{BASE}$. Source: [27]

for b number of bits. M is calculated as follows:

$$M = 2^{b-1} - 1 \tag{3.3}$$

For *Q8BERT*, the weights were quantized to 8-bit, therefore $M = 2^{8-1} - 1 = 127$. The quantization scaling-factor S^x can be calculated based on statistics collected during training, or dynamically determined during inference; in this instance the scaling-factor for weights was calculated as follows:

$$S^W = \frac{M}{max(|W|)} \tag{3.4}$$

The weight quantization was done during training, using a technique called *quantization-aware training*: during model fine-tuning, fake quantization [23] is used to simulate the value errors obtained when rounding down floating-point numbers. These values are then back-propagated to the model, which ends up learning how to overcome quantization errors. To compare the effectiveness between quantization-aware training and simply performing quantization after fully training the model – also called *dynamic quantization* – testing was done on several NLP tasks. The results (displayed in Table 3.1) show that quantization-aware training outperforms dynamic quantization in every task, and even performs better than or equal to the original uncompressed model in certain tasks.

3.2.2 DistilBERT

To lower the memory footprint, an immediate solution would be to train a smaller model; this is one of the main reasons why knowledge distillation has been one of the most researched topics in model compression in recent years. A general-purpose distilled version of *BERT*, *DistilBERT* [7], was developed by teaching a smaller version of the base model using the available pre-trained one as a teacher. It manages to retain 97% of the accuracy from the original model, while being 40% smaller and 60% faster.

Model	Parameters (millions)	Inference time (seconds)	GLUE score	IMDb (acc.)	SQuAD (EM/F1)
BERT$_{BASE}$	110	668	79.5	93.46	81.2 / 88.5
DistilBERT	66	410	77.0	92.82	77.7 / 85.8

Table 3.2: Overall comparison between the pre-trained *BERT* and *DistilBERT*, displaying total number of parameters, inference time of a full pass of a *GLUE* task (sentiment analysis task), resulting score in the *GLUE* benchmark, and performance on two downstream tasks: sentence classification (*IMDB* dataset, measured by accuracy) and question answering (*SQuAD*, measured by ExactMatch (EM) and F1 score). Data source: [7]

DistilBERT has the same architecture as *BERT*, with a focus on reducing the amount of layers from the regular 12 to just 6; the authors also investigated compressing the hidden size dimension, but found that it had a much smaller impact on computation efficiency than compressing the number of layers. Additionally, due to the common dimensionality between the teacher and the student models, the small model was able to be initialized by directly taking layers out of the teacher model.

To train the student model, the authors used a linear combination of the regular supervised training loss with the model distillation loss, which was then back-propagated. The distillation loss over the probabilities of the results output by the teacher model were calculated as follows:

$$L_{ce} = \sum_i t_i * log(s_i) \tag{3.5}$$

where t_i is the probability calculated by the teacher for the soft label i (and likewise, s_i refers to the probability output by the student). These probabilities are calculated using a softmax-temperature [15]:

$$p_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \tag{3.6}$$

where z_i is the score attributed to a class i, and T is the temperature which controls how smooth the output distribution is. During training, the same temperature value was used for both student and teacher probabilities; at inference time T was set to 1, thus becoming a regular softmax function.

The distilled model displayed results expected from following a compression technique (displayed in Table 3.2): the size of the model was considerably reduced and the execution speed rose, while only losing a small amount of performance relative to the base model. The authors also refer to the orthogonality of other model compression techniques relative to knowledge distillation; with additional pruning and quantization, *DistilBERT* could be compacted even further, not without some expected losses in performance.

Additionally, the authors studied the performance of the compact model on mobile devices; two smartphone applications were built for question answering, one using *BERT*$_{BASE}$ and the other *DistilBERT*, and both were deployed on an *iPhone 7 Plus*. The average inference time was measured, with *DistilBERT* being 71% faster than the base model.

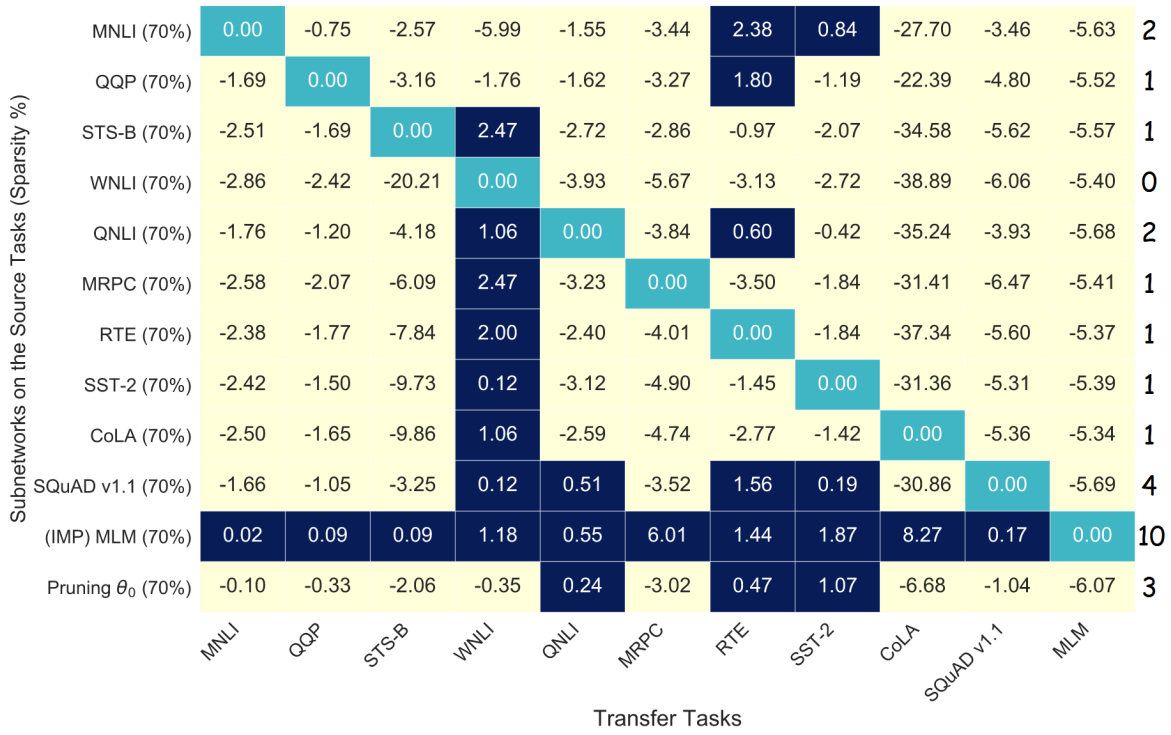


Figure 3.1: The relative performance of subnetworks at 70% sparsity obtained by *iterative magnitude pruning* when transferred to other tasks. Rows represent source tasks and columns represent the target tasks (the task Θ_0 represents the pre-trained *BERT* model, with no further training). Each cell displays the performance of the transferred network for the target task relative to the performance of the pruned network on the same task. **Dark cells** mean the subnetwork found by pruning a model for the source task accomplished a better performance than the subnetwork found for the same target task. Source: [28]

3.3 Lottery Ticket Hypothesis in NLP

In recent years the NLP community focused on building larger Transformer models, as stated before. Concurrently, the computer vision community of researchers explored the Lottery Ticket Hypothesis [29], which states that *"dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that – when trained in isolation – reach test accuracy comparable to the original network in a similar number of iterations"*; simply put, conventional pruning techniques can unveil smaller neural networks (mentioned as subnetworks) which can be trained to reach performances similar to the parent network.

Based on this formulation, and considering the possibility of application towards NLP and its ever-growing models, several concurrent studies [22, 28, 30] focused on applying the same line of thought to *BERT*. The results demonstrate that the Lottery Ticket Hypothesis holds true for NLP, and valuable conclusions arrived from this research:

- Matching "winning ticket" subnetworks can be found between varying values of sparsity, from as low as 40% to as high as 90%. This means that, for specific tasks, a model consisting of roughly

one tenth of the pre-trained *BERT* model can hit similar performances;

- These subnetworks can be found with no extra training required, by directly pruning the pre-trained *BERT* model with no need for any fine-tuning beforehand. The pruned model can still reach a similar performance to the full model;

- For most models fine-tuned to accomplish downstream NLP tasks, the subnetworks found appear to be specific for that specific task, and are unable to be transferred to other tasks.

One of the conclusions opened up new possibilities: "winning tickets" found at 70% sparsity using the task originally used for pre-training *BERT* (masked language modeling) were shown to be universal (results displayed in Figure 3.1), showing that learning can be transferred to other downstream tasks while maintaining accuracy. The resulting implication of a pre-trained *BERT* model properly pruned down to nearly a third of its size still being able to accomplish similar accuracy values when fine-tuned to a downstream NLP task is an incredible breakthrough, especially for low-end or otherwise budget-restricted hardware.

Given the positive results, proper pruning after the initial training could be seen as a second stage of the *BERT* pre-training process in the future. To compare these beneficial improvements in accuracy, and the eventual speedup in inference time, we will be applying *iterative magnitude pruning* (IMP) [28] on our proposed model training experiments.

3.4 Energy Consumption and Carbon Footprint

With larger and better performing models, comes greater resource expenditure. While most NLP models from a decade ago could be properly trained on end-user laptops and other common hardware, training a state-of-the-art model nowadays requires dedicated hardware with several GPUs or TPUs, even if the goal is just to fine-tune an already existing pre-trained model. With this in mind, and considering that properly training and validating a model requires many executions to experiment with different architectures and hyperparameter configurations, the energy consumption is an important (but often forgotten) evaluation detail when training a model. Reporting this detail would allow for cost-benefit analysis between different models, especially when the model is meant to be retrained for downstream usage, such as fine-tuning for a new NLP task.

The amount of energy consumed by a model during the entire training process is not only important in terms of monetary cost, but also due to the effect it has on the environment. The European Environment Agency has reported that, on average, for every kilowatt produced per hour, the equivalent of $275g$ of CO_2 is released to the environment as greenhouse gases [31]. However, since the most popular cloud compute service – Amazon Web Services – is hosted in the United States, it is more reasonable to consider the value reported by the U.S. Environmental Protection Agency (EPA) as the average greenhouse gas emission for model energy consumption: $947lbs$ per megawatt-hour [32], or $430g$ per kilowatt-hour. Considering that data centers spend a substantial amount of energy maintaining servers

Consumption	CO_2e (kg)
Air travel, 1 person, NY↔SF	900
Human life, avg, 1 year	5,000
American life, avg, 1 year	16,400
Car, avg incl. fuel, 1 lifetime	57,153
Training one model (GPU)	
NLP pipeline (parsing, SRL)	18
w/ tuning & experiments	35,592
Transformer (big)	87
w/ neural arch. search	284,019

Table 3.3: Common CO_2 emissions from regular human activity, compared against the training of common NLP models. Source: [10] (converted from lbs to kg)

up and running, and taking in consideration the impact of greenhouse gas emission on the environment, reporting energy consumption is an ever-increasing necessity for any trained models nowadays.

Reporting the power consumption of a model can be done by measuring the overall energy expenditure during training and calculating as follows:

$$p_t = \frac{1.59t \cdot (p_c + p_r + g \cdot p_g)}{1000} \tag{3.7}$$

where p_c is the average power draw (in watts) from all CPU sockets during training, p_r is the average power draw from all DRAM (main memory) sockets, p_g is the average power draw from a GPU during training, and g is the number of GPUs used during training. The estimate power consumption of a model during training is estimated as the combined power draw of CPU, DRAM and GPUs, multiplied by both the number of hours t spent training the model and the Power Usage Effectiveness (PUE) constant, which accounts for the additional energy required for the infrastructure (for data centers, this value mostly accounts for cooling). We consider a PUE coefficient of 1.59, which is the 2020 global average for data centers [33].

To estimate the amount of CO_2 emitted by the model (in kg), we use the value reported by the U.S. EPA depicted above:

$$CO_2e = 0.430 \cdot p_t \tag{3.8}$$

We will be reporting the energy consumption for every model we train, as well as the estimated CO_2 emissions (using the European constant), not only to compare the power usage of an average model against a compressed model, but also as an effort to contribute to the awareness of the environmental footprint of deep learning in NLP.

3.5 Summary

Comparing the final evaluation metric scores against the results of a baseline or state-of-the-art is often not a realistic depiction of the performance of the model; paper authors should always report every experiment detail, such as hyperparameter values and time spent training. Additionally, *expected*

validation performance is a technique that displays the performance of a model based on the results obtained for every training variation, such as different evaluation runs, training time or hyperparameter searches; this expected value better represents the performance of a model, instead of simply showing the best value obtained by the model.

Model compression in NLP has been previously studied, with promising results. On the topic of quantization, *Q8BERT* was developed by applying quantization-aware training to *BERT* and quantizing over 99% of the weights present in the model, obtaining a model that is $4\times$ smaller, with a loss of accuracy of less than 1%. For knowledge distillation, *DistilBERT* was created as a general-purpose distilled version of *BERT*, managing to lose only 3% of the accuracy of the original model, while achieving a 40% size reduction and faster inference time on both common training hardware (60% speed improvement) and low-end hardware (71% speed improvement). Regarding pruning, research conducted on the Lottery Ticket Hypothesis applied to *BERT* found that, for certain downstream NLP tasks, models could be pruned to a sparsity as high as 90% before fine-tuning without losing accuracy, and a more moderate 70% sparsity could universally maintain or even improve accuracy for all tasks (a total of 11 tasks were tested).

Complementing the model evaluation techniques, by obtaining the average power draw of the model during training, the amount of CO_2 emitted by the model (were it to be trained in a data center) can be estimated, thus obtaining a valuable metric to determine the environmental footprint of the model, something often overlooked but very detrimental to the environment.

Chapter 4

Experiments

In this chapter, we describe our approach to evaluate the effects of model compression, and detail our experiments on applying the compression techniques described to different NLP tasks.

4.1 Training Setup

In order to study model compression and its effect on current language models, we picked three fundamental and widely used tasks in NLP:

- Text classification (in particular, sentiment analysis);
- Sequence labeling (specifically, named entity recognition);
- Sentence generation (in this case, dialog-driven generation).

Across every model trained, we used the PyTorch framework [34]. For the models themselves, we picked *BERT*$_{BASE}$ (L=12, H=768, A=12, 110M parameters, 440MB size) [2] as the pre-trained base model to accomplish the sentiment analysis[1] and named entity recognition[2] NLP tasks, and *GPT-2*$_{Small}$ (L=12, H=768, A=12, 117M parameters, 650MB size) [3] for conversation-driven sentence generation[3].

For each task, we planned on performing the following model compression techniques:

- **Baseline:** To obtain the baseline from which we can compare the compressed models, we will fine-tune the pre-trained model for the task at hand.
- **Distillation:** To be able to distill knowledge, we will create a small student model using only the first k Transformer layers from the pre-trained model (*BERT*$_k$). Afterwards, the previously fine-tuned uncompressed model (used as baseline) will act as the teacher model and perform Patient Knowledge Distillation (PKD) (briefly explained in Section 2.1).
- **Pruning:** Following the Lottery Ticket Hypothesis shown in Section 3.3, we will be pruning the pre-trained model directly by gradually applying *iterative magnitude pruning*, and then continue

[1] https://github.com/Uziskull/lightning-text-classification
[2] https://github.com/Uziskull/BERT-NER
[3] https://github.com/Uziskull/lightning-convai

training to recover pruning accuracy losses; from here, the model can be fine-tuned according to the desired task.

- **Quantization:** To perform model quantization, we will use *quantization-aware training* (detailed in Section 3.2.1) during fine-tuning of the pre-trained model for the given task.
- **Quantization + Pruning:** Taking advantage of the compatibility between both compression techniques (as mentioned in Section 2.3), we will be using *quantization-aware training* on the previously pruned model during fine-tuning of the task.

Due to the large amount of training permutations necessary, we approached our testing plan from an iterative standpoint, focusing on applying a given compression technique to every task before moving on to another technique.

4.1.1 Datasets

To train the language models fine-tuned on the sentiment analysis task, we used the *IMDB review* dataset [35] – a group of positive and negative *IMDB* movie reviews. For the named entity recognition task, we made use of the *CoNLL-2003* dataset [36] – a collection of phrases where every word has a corresponding part-of-speech tag, syntactic tag and named entity tag.

```
Sentiment: positive
Review: One of the other reviewers has mentioned
that after watching just 1 Oz episode you'll
be hooked. The...
```

Listing 4.1: Excerpt and sentiment in a review from the *IMDB* dataset.

Word	POS Tag	Syntactic Tag	Entity Tag
U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	.	O	O

Listing 4.2: Word tag examples from the *CoNLL-2003* dataset, displayed in columns with labeling above.

To train and fine-tune the *GPT-2* based conversational model on the task of sentence generation, we used the *Persona-Chat* dataset [37], which is a crowd-sourced collection of over 160 thousand utterances between pairs of personas:

```
I love the beach.
My dad has a car dealership.
I just got my nails done.
I am on a diet now.
Horses are my favorite animal.
```

Listing 4.3: Example personas from the *Persona-Chat* dataset.

Additionally, to prune both *BERT* and *GPT-2* based models, we need to pre-train the models on the NLP tasks of masked language modeling and causal language modeling, respectively; we made use of the *WikiText* corpus [38], a bundle of several million tokens extracted from verified articles on *Wikipedia*:

```
= Super Mario Land =

Super Mario Land is a 1989 side @-@ scrolling
platform video game , the first in the Super Mario
Land series , developed and published by Nintendo
as a launch title for their Game Boy handheld
game console .
```

Listing 4.4: Excerpt from an article in the *WikiText-2* dataset.

4.1.2 Evaluation

We followed a quantitative approach to evaluation, focusing on comparing the original models to their compressed counterparts. Following a set of guidelines for best practices [26], while also measuring some compression-related details, we reported the following details for every model trained and evaluated:

- A description of the computing infrastructure used during training;
- The average runtime for each approach;
- Details of train/validation/test splits;
- Corresponding validation performance for each reported test result;
- A link to the implemented code;
- The response time of the model during execution;
- The size of the model.

NLP Tasks	GPU	Data Splits (train/dev/test)	Train Epochs	Batch Size (train/test)	Learning Rate
Sentiment Analysis	GeForce GTX 1080 Ti (11GB)	50/25/25	5	8/8	{3.0e-05, 1.0e-04, 3.0e-04, 5.0e-05}
Named Entity Recognition	GeForce GTX TITAN X (12GB)	70/15/15	5	32/8	{3.0e-05, 1.0e-04, 3.0e-04, 5.0e-05}
Sentence Generation	GeForce GTX 1080 Ti (11GB)	99/0.5/0.5	3	2/1	{6.25e-05, 5.0e-05, 1.0e-04}

Table 4.1: General training details for every task, including hardware and hyperparameter configurations. All settings remained the same across every model trained in said task, whether compressed or not.

To measure the performance of each model, we made use of conventional accuracy metrics commonly used for evaluating both *BERT* and *GPT-2* based models. For sentence generation, we used *BLEU*, *TER*[4], *BERTScore*, and *Hits@1/5/10*. For sentiment analysis and named entity recognition, we calculated *accuracy*, *precision*, *recall* and *F1 score*.

These values were measured using validation data obtained during model testing, and presented via *expected validation performance* [26] of the model as a function of computational budget. By obtaining our results via this technique and subsequently presenting them on graphs, a better comparison between the original models and their compressed versions can be established, as accuracy values will be compared in relation to the computational budget provided.

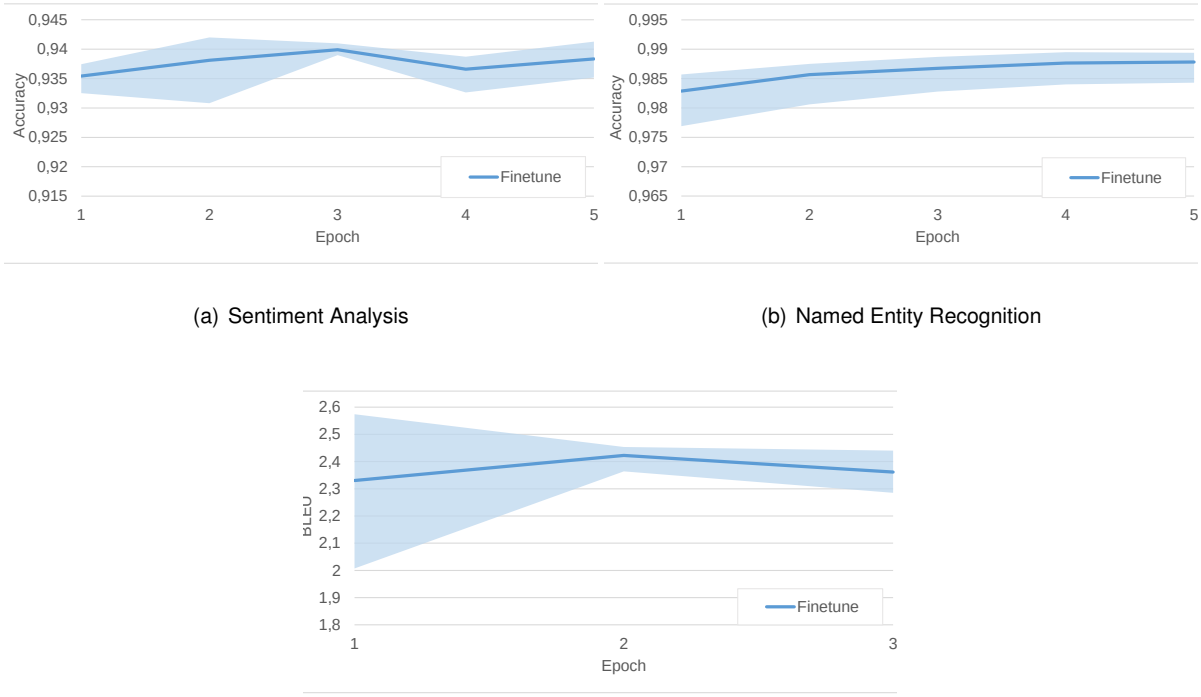
Additionally, we used heuristics related to energy consumption and environmental footprint [10] to compare the budget required to train a model against its compressed alternative. For every model trained, the average GPU power draw was obtained to calculate the power consumption of the model, which will then be used to estimate the amount of CO_2 produced (as explained in Section 3.4). While it is not the only component consuming energy, we will only be focusing on the GPU power draw as it is the main power funnel when training a model.

4.2 Testing Details

In order to properly compare the model compression techniques, we set up a controlled testing environment by maintaining the same testing hardware and hyperparameters across models for the same tasks. Every model was trained on a cloud computing environment, using a dedicated GPU for the duration of the training process; all of these testing configurations are presented in Table 4.1.

To train each model, a script was used to load and start the training session. The framework used to train the models accurately took note of the evaluation details, such as the time it took to train and evaluate each model and the metrics measured during inference. Additionally, to measure the average GPU power draw for every model trained, the script ran a constant query to the NVIDIA System Management Interface (`nvidia-smi`[5]) in parallel to the model training itself, which gathered and registered

[4]Unlike the other metrics shown here, a lower TER score is preferable to a higher score.
[5]`https://developer.nvidia.com/nvidia-system-management-interface`



(a) Sentiment Analysis

(b) Named Entity Recognition

(c) Dialogue-driven Sentence Generation

Figure 4.1: Experiment results from fine-tuning the task-specific models, to be used as baseline comparisons for the compressed models.

the current power draw of the GPU being used, with an interval of 5 seconds; those values were then used to calculate the overall average power draw.

Before delving into any experiments using model compression, we began our testing by fine-tuning every model with no compression technique applied, such that we could use the results as a baseline comparison for the compressed models. Each model was trained using the same hyperparameters, while only varying the learning rate. In order to keep testing coherency, we made every run deterministic by setting a fixed seed for randomness, which prevented variations in weight initialization and data order across runs; this was done to ensure that, throughout our whole experiment, the number of variable details within our control was reduced to just the different learning rates and the different compression techniques applied, such that the results obtained could be more focused on the model compression itself. This decision to constrain hyperparameter search meant that our fine-tuned models did not obtain the best results reported by the authors of the different models, however this was outside the scope of our work since our focus was set on the effects of compression techniques regarding the model performance, regardless of the ranking of the model. The results obtained from fine-tuning the models on their given tasks are displayed in Figure 4.1 (the complete evaluation metrics are displayed in Appendix A).

NLP Tasks	Pre-trained Teacher		PKD Student	
	Base Model	Weight Initialization	Base Model	Weight Initialization
Sentiment Analysis Named Entity Recognition	*BERT*$_{12}$ (12 layers, 440MB)	Pre-trained weights	*BERT*$_{6}$ (6 layers, 270MB)	6 first layers of pre-trained weights
Sentence Generation	*GPT-2*$_{12}$ (12 layers, 650MB)	Pre-trained weights	*GPT-2*$_{6}$ (6 layers, 480MB)	6 first layers of pre-trained weights

Table 4.2: Student models created for *patient knowledge distillation* and their initialization weights, compared to the models used for the tasks.

4.3 Knowledge Distillation

Unlike the other compression techniques used in this work, knowledge distillation cannot be directly applied on a pre-trained model to generate a new, compressed model. Instead, the compression factor comes from the usage of a smaller student model, and the distillation process serves as a tool to transfer information from the trained teacher model to the student model. As such, to compress each task-specific model, we designated the fine-tuned baseline model as the teacher and created a smaller version of the model to be used as the student; following the authors of the paper regarding PKD [16], the network architecture of the smaller model is identical to the base model used for the task, but the number of hidden layers was cut in half and only some of the pre-trained weights were used for initialization (the differences between the teacher and student models are outlined in Table 4.2).

For both the sentiment analysis and named entity recognition tasks (the ones that made use of the base *BERT* model), to transfer the knowledge from the teacher model to the target student model using PKD, we took the entire training dataset used for the student model (which in our experiments is the same as the one used for fine-tuning the baseline model) and ran it through the teacher model; for every element in the training data, the fine-tuned model processed the input and returned the values resulting from the output layer, as well as the values within the intermediate model layers. Every set of these values was then saved as extra knowledge, and was integrated with the training dataset of the student model such that, when training the student, the training loss of the model got calculated based on the regular fine-tuning loss, the distillation loss over the predictions of the teacher model, and the mean-square loss of the intermediate layer values. Since the teacher model has 12 layers while the student model only has half of those, we had to pick the set of intermediate layers to distill from (denoted as I_{pt}); we followed the PKD-Skip strategy [16] and made use of the layers $I_{pt} = \{2, 4, 6, 8, 10\}$, which together with the output layer of the teacher model resulted in the 6 layers that made up the student model. The set I_{pt} was the same used throughout every model we applied PKD to. The performance of the student model and the comparison with the baseline model is displayed in Section 5.1.

We were unable to perform the same teacher knowledge gathering for the sentence generation task (the one that made use of the base *GPT-2* model), since the resulting dataset with extra knowledge was too large to be able to be loaded during student training; this occurred due to *Persona-Chat* [37], the

IMP Parameters	Batch Size	MLM Loss Probability	Block Size	Learning Rate	Noise Rate	Adam ϵ	Pruning Step	GPU
BERT	4	15%	512	5.0e-5	0.01	1.0e-8	10%	GeForce GTX TITAN X
GPT-2	1	N/A	1024					

Table 4.3: Training details used for training over the pre-trained models, using iterative magnitude pruning.

WikiText Comparison (on BERT)	IMP to 70% sparsity, with 10% magnitude pruning step every Y iteration					
	Y = 5000		Y = 10000		Y = 15000	
	Accuracy	Perplexity	Accuracy	Perplexity	Accuracy	Perplexity
WikiText-2	59.5332	9.0276	59.755	9.6268	59.4877	10.5135
WikiText-103	61.8877	7.147	63.7721	6.2961	64.4464	5.8791

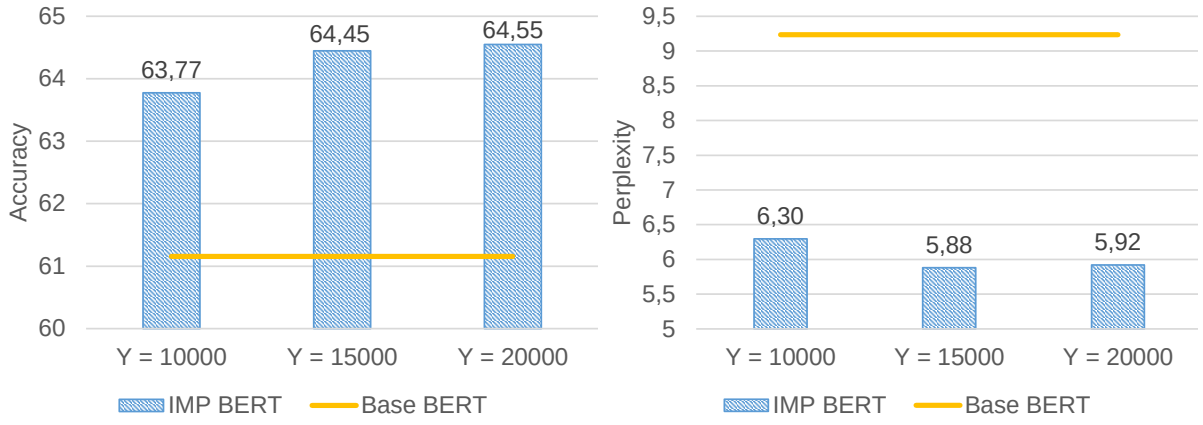
Table 4.4: Comparison between *WikiText-2* and *WikiText-103* when used to train a pre-trained *BERT* model while applying iterative magnitude pruning.

dataset used for fine-tuning this task, being much bigger than the datasets used to train the *BERT* based models. To get around this issue, we forwent the knowledge gathering before training the student model, and calculated the necessary intermediate and final values of the teacher model on-the-fly instead; we loaded both models at the same time on the setup of every run, and during the main training loop the input was firstly fed to the teacher model to get the necessary resulting values, which were then used for the loss calculation of the student model. As detailed before, the same set of intermediate layers I_{pt} was used for this training. Although the end result is the same as gathering the extra information beforehand, loading two models at the same time had its drawbacks, which we discuss in greater detail in Section 5.1 along with the results of the PKD student compared to the baseline model.

4.4 Pruning

To prune every task-specific model, we first took the pre-trained base model for each task and trained it further while applying IMP [28], which prunes the attention heads of the model over several training epochs. IMP is applied as follows: during model training, every time a certain number of training iterations complete, the weights of the model are pruned by a specific incremental percentage; the weights are then reset to their original values, and the training continues, up until the total sparsity of the model reaches the defined goal; all of these values (the number of iterations, the prune step and the desired final sparsity) are defined beforehand. The original code supplied by the paper authors only worked for *BERT* models, thus some changes were made to the code to also accept and prune *GPT-2*. All the parameters used for IMP training are displayed in Table 4.3.

As previously mentioned, we decided to use the *WikiText* [38] corpus for the required extra training on the pre-trained base models since it contains large amounts of curated text from *Wikipedia*, which is ideal for masked language modeling (the task on which *BERT* is trained) and causal language modeling



(a) Accuracy

(b) Perplexity

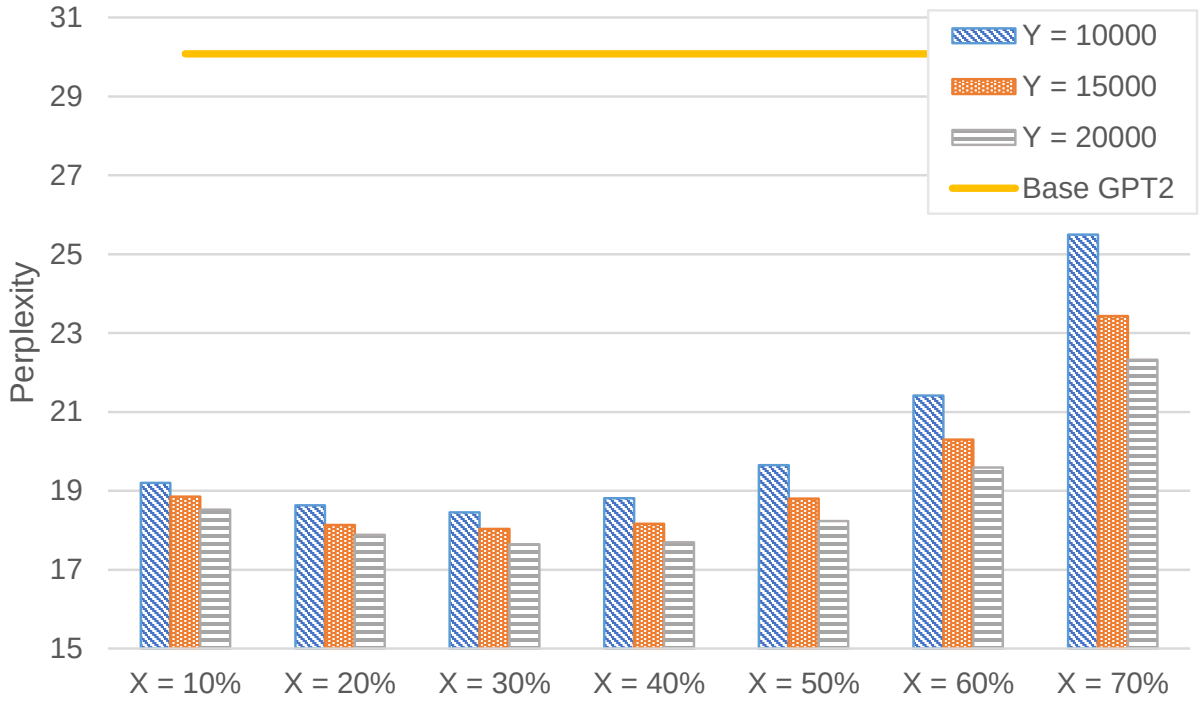
IMP Model	Training Time (s)	Power Spent (W/h)	CO_2 Emitted (kg)
Y=10000	35890	2154,89	0,94
Y=15000	53800	3232,47	1,41
Y=20000	71740	4332,14	1,89

(c) Training time, power usage and estimated CO_2 emissions

Figure 4.2: Experiment results for *BERT* models after applying iterative magnitude pruning to 70% sparsity, by pruning 10% of the weights every Y training steps. The models were trained on the *WikiText-103* dataset. The accuracy and perplexity of the base *BERT* model is displayed for comparison.

(the task on which *GPT-2* is trained). There are two *WikiText* corpora available for usage – *WikiText-2* and *WikiText-103* – which contain 2 million tokens and 103 million tokens, respectively. To understand which of these corpora would result in a better performing model, we conducted a small comparison between both on a pre-trained *BERT*: we applied IMP to 70% sparsity with a 10% pruning step every Y iterations, varying between three different numbers of iterations. The results (detailed in Table 4.4) show that models trained and pruned using the *WikiText-103* corpus always obtained higher accuracy and lower perplexity scores, so we decided to use that corpus for the remainder of the models trained during IMP.

With the corpus picked, we began applying IMP to the pre-trained *BERT* model. According to the experiments done by the paper authors (as detailed in Figure 3.1), a *BERT* model trained on masked language modeling with 70% sparsity can be transferred to other downstream tasks and achieve the same performance (or in some cases even outperform said task), compared to pruning the downstream task directly. Given this, we decided to prune *BERT* to 70% sparsity with a 10% pruning step every Y iterations, varying the number of iterations to find which model would display better results overall; the results obtained from training, along with other evaluation metrics, can be seen in Figure 4.2. Interpreting these results, it is important to note that the main reason for the difference in accuracy and perplexity between the pruned models and the base *BERT* model is the training data in use: *BERT* was pre-trained using both BookCorpus [39] and the English Wikipedia, while our models were trained with only an excerpt of the full English Wikipedia corpus (*WikiText-103*). Therefore, although both models are trained towards the same masked language modeling task, comparing these values is erroneous and a proper comparison between results should only be done between the fine-tuned version of the models,



(a) Perplexity

IMP Model	Training Time (s)	Power Spent (W/h)	CO_2 Emitted (kg)
Y=10000	11690	695,08	0,30
Y=15000	17542	1056,06	0,46
Y=20000	23381	1419,01	0,62

(b) Training time, power usage and estimated CO_2 emissions (for 30% sparsity)

Figure 4.3: Experiment results for *GPT-2* models after applying iterative magnitude pruning to $X\%$ sparsity, by pruning 10% of the weights every Y training steps. The models were trained on the *WikiText-103* dataset. The perplexity of the base *GPT-2* model is displayed for comparison.

since the same dataset will be used.

The research conducted on the paper applied IMP solely to *BERT* based models, and did not focus on any other transformer models. This meant that the universality of task transferal observed at 70% sparsity for *BERT* could not be directly inferred to the *GPT-2* model, since that would be a baseless assumption. As such, we decided to conduct some extra testing to observe what the ideal sparsity for the model should be, in order to achieve a proper size reduction while keeping the model performance as unaffected as possible. To test this, we began applying IMP to the pre-trained *GPT-2* model using the *WikiText-103* corpus, pruning the model to $X\%$ sparsity with a 10% pruning step every Y iterations, varying both the number of iterations and the sparsity itself (between 10% and 70%); the relevant results are displayed in Figure 4.3 (the complete test results are displayed in Appendix A).

As with the *BERT* models, the difference in perplexity seen between the base *GPT-2* model and the pruned ones is mainly due to a different dataset being used to apply IMP. More importantly, a trend can be seen throughout all of the different sparsity percentage models: the resulting perplexity is lowered until the model becomes 30% sparse, then it steadily begins to rise as the sparsity grows. We attribute this behavior to the number of weights that get pruned and become unused within the model, which

gradually make the model less knowledgeable in regards to predicting the following masked token. Additionally, since a higher level of sparsity requires more training time performing IMP, the perplexity also increases due to the model getting used to the dataset and over-fitting the training data. Following this line of thought, we chose 30% sparsity as a valid sparsity percentage for pruning *GPT-2*, and focused on comparing the results from the different training iterations used when performing IMP at that sparsity.

With both base models successfully pruned, we had to pick the best models in order to fine-tune them. The accuracy reported on the results of each pruned model refers to the tasks of masked language modeling in *BERT* based models and causal language modeling in *GPT-2* based models, which are both different tasks from the final ones the models are going to be fine-tuned in. We decided to use the perplexity metric to decide which model was better, since it is an evaluation method intrinsic to the model itself and does not directly evaluate the task at hand; this resulted in the two chosen models being *BERT* pruned to 70% sparsity with a 10% pruning step every 15000 iterations, and *GPT-2* pruned to 30% sparsity with a 10% pruning step every 20000 iterations. These models were then fine-tuned in the same manner as the baseline models; the results and the comparison between the pruned and baseline models are displayed in Section 5.2.

4.5 Quantization

Given the positive results of previously quantized models (such as the one detailed in Section 3.2.1), our initial approach to apply and study quantization was to make usage of quantization-aware training. This also aligned with our goal of studying easy-to-use compression techniques, since QAT is made available as a simple set of functions within the PyTorch framework. However, when we began experimenting with applying QAT to *BERT* and *GPT-2* models, despite the models being able to be trained with this compression setup and properly simulating the effects of quantization, the technique always failed at the final steps of creating the model; at the time of writing, PyTorch does not directly support applying QAT to models that require embedding layers, thus ruling out QAT on Transformer architecture models such as *BERT* and *GPT-2*. While solutions to this problem exist, all of them require fundamental changes to the architecture of the model, which in turn make QAT a non-versatile compression technique to apply and place it outside of the scope of this work; with this in mind, we decided to study an easy-to-use quantization technique instead, Dynamic Quantization (DQ).

To quantize every model, we applied DQ to the fine-tuned models used as baseline. Being a post-training quantization technique, DQ requires no modification to the training setup nor additional training time, so the compression process was very straightforward; we loaded the fine-tuned models, applied DQ to quantize the weights of the models to 8-bit (using the *QNNPACK*[6] backend) and evaluated them. The resulting performance of every model, as well as the comparison with the baseline, are displayed in Section 5.3.

Although DQ was a simple compression technique to apply, we had to perform some changes in order to use it on the *GPT-2* based model. The quantization backend used by DQ only supports a select

[6] `https://github.com/pytorch/QNNPACK`

few common neural network layers, such as the Linear layer (which applies a linear transformation to the incoming data) used by *BERT*; however, *GPT-2* based models make use of the Conv1D layer (which applies a one-dimensional convolution over the incoming data), which is not supported by the backend. In order to be able to quantize *GPT-2*, every Conv1D layer had to be swapped with Linear layers after loading the model; despite being a change in the framework of the model, this was done on-the-fly before quantizing the model with very low loading time penalty and can be considered an extra step in applying DQ. While this change allowed for *GPT-2* models to be quantized, it had some negative effects on the performance of those models, which we discuss in greater detail in Section 5.3.

Additionally, to evaluate the performance of compressed models on low-end hardware, we ran both the baseline and the quantized models on a *Raspberry Pi 4*, a compact and modular single-board computer designed for portability and flexibility of use (the model we used has 4GB of RAM), to compare the inference time between the executed models, as well as between the models ran on the *Raspberry Pi* CPU versus the ones ran on the cloud computing environment with the dedicated GPU. The resulting comparison is detailed over in Section 5.3.

4.6 Quantization + Pruning

To test the combined effects of quantization and pruning, we applied DQ to the models previously pruned via IMP and evaluated the resulting models. As discussed before, due to the nature of DQ, applying it to pruned models was a simple task, although the same workaround had to be done for the pruned *GPT-2* model. Furthermore, the quantized and pruned models were also ran on the *Raspberry Pi* to compare inference times; all of these results are displayed in Section 5.4.

4.7 Summary

We evaluated three different tasks – sentiment analysis, named entity recognition and dialog-driven sentence generation – against three different model compression techniques – *patient knowledge distillation* (PKD), *iterative magnitude pruning* (IMP) and *dynamic quantization* (DQ) (additionally, we also evaluated the combination of pruning and quantization) – and reported every detail regarding the testing environment used.

For PKD, we ran the training dataset through the teacher model and gathered the results for the intermediate model layers, which were then used to train the student model, thus achieving distillation of knowledge. This was not possible to do for the sentence generation task due to the size of the dataset; to get around this, the teacher model was loaded at the same time as the student model during its training, and the results were calculated on-the-fly.

For IMP, we experimented with the pruning of the pre-trained model, applying 70% sparsity to *BERT* models with varying training steps per pruning. Since the original paper referred to *BERT* models only, to prune the *GPT-2* models we experimented with different sparsity values and concluded that 30% sparsity was the ideal value.

For quantization, we initially experimented with applying quantization-aware training, but found out that PyTorch does not support this quantization method on models that require embedding layers, which is the case for Transformer-based models like *BERT* and *GPT-2*. Therefore, we decided to experiment and evaluate models quantized with DQ, which was directly applied to the previously fine-tuned baselines without requiring any extra training whatsoever. DQ was also applied to the pruned models after fine-tuning them, to study the effects of combining both compression techniques.

Chapter 5

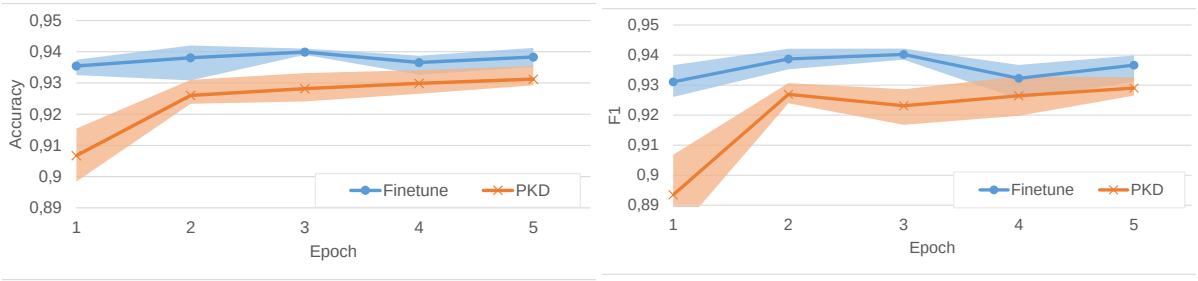
Results

In this chapter, we review the results of the experiments described in Chapter 4 and establish comparisons to the baseline models in terms of performance and resource usage metrics.

The results shown focus on the most relevant metrics for each model (accuracy and F1 score for *BERT* models, and BLEU and TER for *GPT-2* models); the full results of the recorded metrics can be seen in the Appendix Table A.1. Unless specified otherwise, the training time and power spent refer to the full model training process, the inference time refers to the time the model took to perform an entire run over the evaluation dataset (according to the training details in Section 4.2), and the estimated CO_2 emissions refer to all the runs performed for that task.

5.1 Knowledge Distillation

Overall, knowledge distillation had an outstanding result on *BERT* based models. For the sentiment analysis task (displayed in Figure 5.1), none of the recorded metrics for the distilled model degraded more than 1% compared to the fine-tuned baseline: the model got an accuracy of 0.931 and a F1 score of 0.929, which is only 0.76% worse than the accuracy of the baseline model (0.938) and 0.82% worse than the F1 score of the baseline model (0.937). For the named entity recognition task (shown in Figure 5.2), the distilled model obtained an accuracy of 0.985 – just 0.27% worse than the accuracy of 0.988 observed in the baseline model – and a F1 score of 0.928, faring 1.38% worse compared to the F1 score of the baseline model (0.941). In both task results (but more noticeable in the named entity recognition task) we can see that the performance ranges between the compressed model and the baseline overlap, and, for the highest scoring learning rates used in training, the model results get very close to the expected performance of the baseline model; this implies that a well-trained distilled model using proper hyperparameter optimizations could outperform the original uncompressed model, although this verification is outside the scope of this work. The small size of these distilled models (270MB, 39% smaller than the original baseline model) resulted in overall shorter training times, which consequently lowered the power spent and the resulting calculated CO_2 emissions. This also influenced the inference times, with the distilled model fine-tuned on the task of sentiment analysis achieving an outstanding 79%



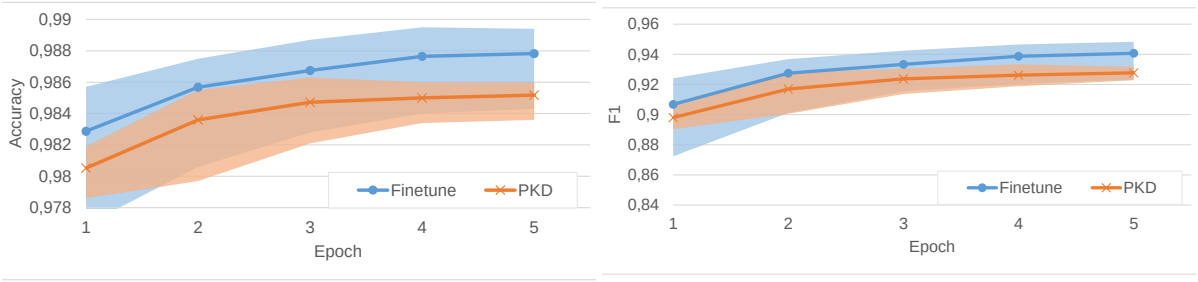
(a) Accuracy

(b) F1 Score

	Model Size (MB)	Avg. Training Time (s)	Avg. Inference Time (s)	Avg. Power Spent (W/h)	Total CO_2 Emitted (kg)
Baseline	440	6289	32.01	361.77	0.63
PKD	270	3084	6.85	186.05	0.33

(c) Model size, average training/inference time and power usage, and estimated CO_2 emissions across all runs

Figure 5.1: Experiment results for distilled models fine-tuned in sentiment analysis.



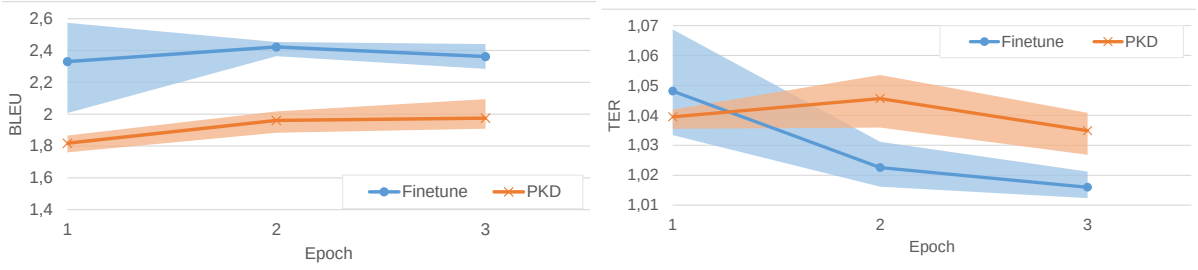
(a) Accuracy

(b) F1 Score

	Model Size (MB)	Avg. Training Time (s)	Avg. Inference Time (s)	Avg. Power Spent (W/h)	Total CO_2 Emitted (kg)
Baseline	440	5372	49.74	235.37	0.41
PKD	270	4582	39.49	192.91	0.34

(c) Model size, average training/inference time and power usage, and estimated CO_2 emissions across all runs

Figure 5.2: Experiment results for distilled models fine-tuned in named entity recognition.



(a) BLEU

(b) TER

	Model Size (MB)	Avg. Training Time (s)	Avg. Inference Time (s)	Avg. Power Spent (W/h)	Total CO_2 Emitted (kg)
Baseline	650	101286	3072.52	6009.83	7.88
PKD	480	153838	1417.95	9290.23	12.19

(c) Model size, average training/inference time and power usage, and estimated CO_2 emissions across all runs

Figure 5.3: Experiment results for distilled models fine-tuned in sentence generation.

improvement over the baseline model, and the named entity recognition model achieving 21% faster inference times.

For the *GPT-2* based models fine-tuned in sentence generation (displayed in Figure 5.3), knowledge distillation proved to be the worst performing model compression technique, with the distilled model achieving a BLEU score of 1.975 – a loss of 16.37% compared to the baseline score of 2.362 – and a TER score of 1.035, a slight increase of 1.86% over the baseline score of 1.016; this drop in model quality was felt throughout all evaluation metrics, which can be seen in the Appendix Table A.1. Similarly to the other tasks, the distilled sentence generation model ended up with a size 26% smaller than the baseline model, which directly affected the lowering of the inference time by a sizable 54% compared to the baseline. However, due to the workaround used to perform PKD (discussed in detail in Section 4.3), there were effectively two different models loaded and executed during training, which greatly extended the training time by 52%; this had a directly proportional impact on the power spent and, consequently, on the estimated CO_2 emissions.

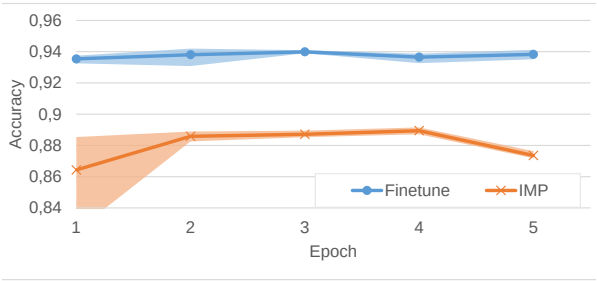
We mostly attribute the large result deterioration of the models fine-tuned in sentence generation to the architecture of the small student model. Considering the observed results we conclude that, for a complex[1] model such as *GPT-2*, a compact version of the same model with only 6 layers does not have enough model parameters to properly learn all the information it needs from the teacher model; however, we believe that the same student model architecture with some more layers (or even a different model architecture altogether) could better learn from the teacher model, such that a proper trade-off between quality and size can still be achieved.

5.2 Pruning

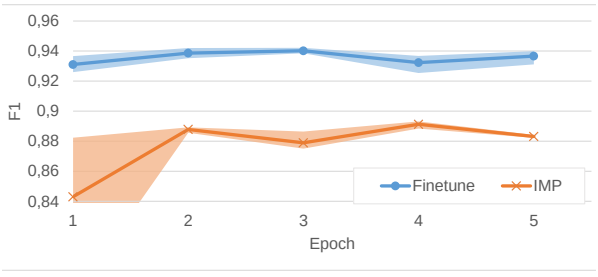
Observing the results, we can see an overall decrease in model performance metrics compared to the baseline model. For the sentiment analysis task (displayed in Figure 5.4), the models pruned using IMP obtained an accuracy of 0.874 and a F1 score of 0.883, which is 6.90% and 5.71% worse than the scores obtained by the baseline, respectively; for the named entity recognition task (displayed in Figure 5.5), the difference is smaller but still negative, with an accuracy of 0.981 (0.68% worse than the baseline) and a F1 score of 0.907 (3.53% worse than the baseline). Additionally, the results for both tasks show no major difference from the baseline models in regards to training time, inference time, and the power spent training and calculated CO_2 emissions thereof.

Interestingly, IMP provided a noticeable improvement for the models fine-tuned on the sentence generation task (shown in Figure 5.6), obtaining a BLEU score of 2.682 – a substantial 13.56% increase over the baseline – and a TER score of 1.024, a small increase of 0.78% compared to the baseline. Additionally, the training and inference times for the pruned model are 4% and 9% smaller than the ones reported for the baseline, respectively; we attribute this difference in timing to the more efficient computation of the complex convolution layers present in *GPT-2* models, given that 30% of the weights

[1] While both *BERT* and *GPT-2* fundamentally have the same Transformer-based architecture [1], *GPT-2* uses more complex "decoder" blocks on its layers while *BERT* uses the simpler "encoder" blocks, hence our classification of complex model.



(a) Accuracy

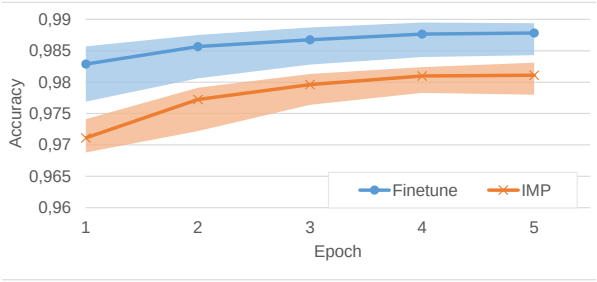


(b) F1 Score

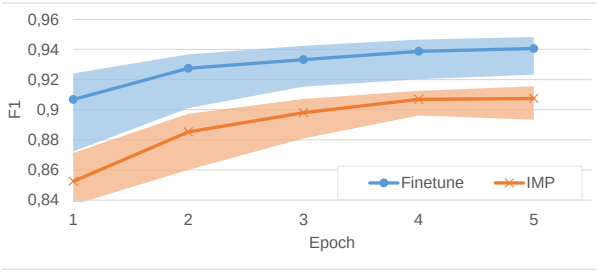
	Model Size (MB)	Avg. Training Time (s)	Avg. Inference Time (s)	Avg. Power Spent (W/h)	Total CO_2 Emitted (kg)
Baseline	440	6289	32.01	361.77	0.63
IMP	440	6368	30.99	361.83	0.63

(c) Model size, average training/inference time and power usage, and estimated CO_2 emissions across all runs

Figure 5.4: Experiment results for pruned models fine-tuned in sentiment analysis.



(a) Accuracy

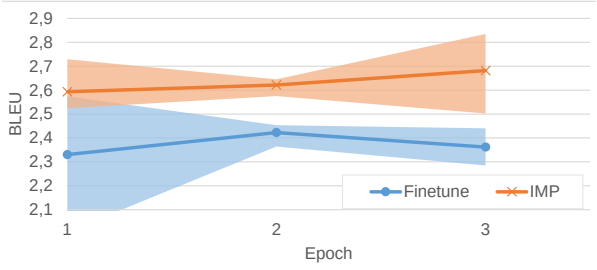


(b) F1 Score

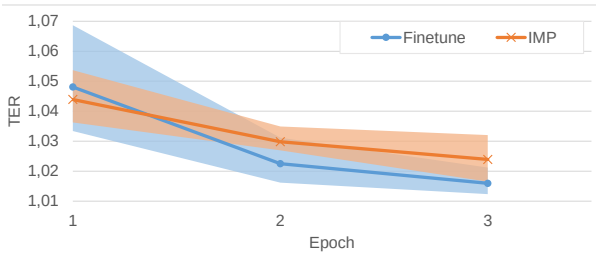
	Model Size (MB)	Avg. Training Time (s)	Avg. Inference Time (s)	Avg. Power Spent (W/h)	Total CO_2 Emitted (kg)
Baseline	440	5372	49.74	235.37	0.41
IMP	440	5347	49.61	235.02	0.41

(c) Model size, average training/inference time and power usage, and estimated CO_2 emissions across all runs

Figure 5.5: Experiment results for pruned models fine-tuned in named entity recognition.



(a) BLEU



(b) TER

	Model Size (MB)	Avg. Training Time (s)	Avg. Inference Time (s)	Avg. Power Spent (W/h)	Total CO_2 Emitted (kg)
Baseline	650	101286	3072.52	6009.83	7.88
IMP	650	97124	2795.51	5786.84	7.59

(c) Model size, average training/inference time and power usage, and estimated CO_2 emissions across all runs

Figure 5.6: Experiment results for pruned models fine-tuned in sentence generation.

	Model Size (MB)	Avg. Inference Time per single example (s)		Accuracy	F1
		CPU	GPU		
Baseline	440	10.000	0.003	0.938	0.937
DQ	175	4.850	0.286	0.934	0.927

Table 5.1: Experiment results for quantized models fine-tuned in sentiment analysis.

	Model Size (MB)	Avg. Inference Time per single example (s)		Accuracy	F1
		CPU	GPU		
Baseline	440	5.373	0.014	0.988	0.941
DQ	175	2.155	0.170	0.979	0.894

Table 5.2: Experiment results for quantized models fine-tuned in named entity recognition.

	Model Size (MB)	Avg. Inference Time per single example (s)		BLEU	TER
		CPU	GPU		
Baseline	650	885.912	3.073	2.362	1.016
DQ	280	326.905	54.526	2.076	1.016

Table 5.3: Experiment results for quantized models fine-tuned in sentence generation.

are set to zero. The lower training time also has the side effect of lowering the amount of power spent during training and, consequently, the estimated CO_2 emissions.

The removal of 30% of the smaller weights within the model led to several improvements on the sentence generation task model. We attribute this gain to the now-removed weights causing less interference on further calculations regarding their respective weight matrices, while not removing too many weights so as not to take a larger hit on model quality; we believe that complex models, like those based on *GPT-2*, have larger numbers of these noisy weights, thus taking better advantage of pruning techniques. However, despite displaying these improvements and only slightly worsening the performance of the remaining models, this model compression technique ultimately failed the main goal of reducing model size; this is due to the problem with representing sparse matrices (briefly discussed in Section 2.2). With only 70% of the weights pruned in *BERT* based models (and an even lower 30% for *GPT-2* based ones), no method of sparse matrix representation would allow for a smaller sized weight storage with no additional detrimental computational overhead [25]; additionally, PyTorch currently offers no way of saving or loading weights as sparse matrices, so we had to store the pruned models with the same representation as the fine-tuned models, thus saving no model size whatsoever. Nevertheless, we conclude that, despite not saving space in practice, magnitude pruning is an effective way of compressing a model while maintaining (and even improving) the quality and performance of the model.

5.3 Quantization

As previously mentioned in Section 4.5, unlike the other compression techniques we experimented with, DQ is applied post-training; therefore, we cannot analyze the expected validation performance development over the training epochs, nor is there any training time or power usage to compare against. Observing the final results, we can see minor quality degradation on the *BERT* based models: for the task of sentiment analysis (displayed in Table 5.1), DQ scored 0.934 in accuracy and 0.927 in F1 score, only 0.51% and 1.00% worse than the respective scores for the baseline model; for the task of named entity recognition (shown in Table 5.2), DQ slightly worsened the accuracy by 0.91% compared to the baseline by obtaining a score of 0.979, and got a F1 score of 0.894, 5.00% worse than the baseline score.

Regarding the sentence generation task, the results (displayed in Table 5.3) show that, compared to the baseline, the quantized fine-tuned model severely worsened its BLEU score by 12.10% with a value of 2.076, while obtaining a TER score of 1.016, a very narrow improvement of 0.01% over the baseline. As previously explained in Section 4.5, this large drop in model quality can be attributed to the architectural change made on-the-fly to be able to apply DQ to *GPT-2* based models; the one-dimensional convolution operations performed over data, present in the layers of the original architecture, are drastically different from the linear transformation operations performed by the replaced layers, which lead to different results from the ones the model was originally trained to obtain. We believe this negative outcome can be reduced if the model architecture is changed before training and quantizing, preferably with a pre-training quantization technique; we were unable to experiment with QAT, and as such leave this hypothesis as future work in this area.

While all quantized models showed significant model size reduction, with the *BERT* based models being compressed to 175MB (60% of the original model size) and the *GPT-2* based models ending up with a size of 280MB (57% of the starting size), this is only reflected in the size loaded in memory; there is no current way of saving or directly loading a quantized model in PyTorch. The model still has to be stored as a non-quantized floating-point model, which is then loaded and quantized after loading, and only then can the quantized integer weights be loaded.

The major improvement with quantized models, however, is their performance in CPU-optimized hardware. Comparing both the baseline and DQ models when ran on the *Raspberry Pi*, operations within the model layers using integer weights prove to be much faster, with the sentiment analysis task executing queries 51.50% faster, the named entity recognition task showing 59.89% lower latency, and the sentence generation task performing 63.10% faster compared to the baseline models. However, it is worth noticing that modern GPUs can still perform integer-based operations and end up running CPU-optimized models faster than a regular CPU [9]; therefore, despite the latency improvement in a CPU environment, quantized models executed on a GPU still outperform their counterparts ran on a CPU: the sentiment analysis task runs 94.10% faster on GPU, the named entity recognition task executes 92.11% faster and the sentence generation task takes 83.32% less time. Lastly, since GPUs are optimized to perform operations on floating-point weights, the time taken by the GPU to execute operations on integer values is directly responsible for DQ models to show exponentially higher latency values: quantized

	Model Size (MB)	Avg. Inference Time per single example (s)		Accuracy	F1
		CPU	GPU		
Baseline	440	10.431	0.003	0.938	0.937
IMP+DQ	175	5.155	0.286	0.883	0.881

Table 5.4: Experiment results for pruned & quantized models fine-tuned in sentiment analysis.

	Model Size (MB)	Avg. Inference Time per single example (s)		Accuracy	F1
		CPU	GPU		
Baseline	440	4.870	0.014	0.988	0.941
IMP+DQ	175	1.917	0.171	0.956	0.775

Table 5.5: Experiment results for pruned & quantized models fine-tuned in named entity recognition.

	Model Size (MB)	Avg. Inference Time per single example (s)		BLEU	TER
		CPU	GPU		
Baseline	650	868.929	3.073	2.362	1.016
IMP+DQ	280	348.607	54.467	2.373	1.009

Table 5.6: Experiment results for pruned & quantized models fine-tuned in sentence generation.

models run 98.95% slower for the sentiment analysis task, 91.76% slower for the named entity recognition task, and 94.36% slower for the sentence generation task. From these results, we can conclude that quantization is a valuable compression method to apply on models deployed on low-end CPU-optimized hardware, but its usage is highly detrimental on hardware environments that make use of GPUs.

5.4 Quantization + Pruning

By quantizing the previously pruned models, we obtained a better insight into the effects of quantization on language models. Much like the conclusions we inferred from the results of the quantized models, applying DQ on models pruned with IMP resulted in a smaller model size and faster inference times when executed on CPU, leading to further reinforcement of our findings regarding performance on low-end hardware. However, compared to the aforementioned quantization results, we can observe an interesting effect on the model quality depending on the results previously obtained from the pruned models for every NLP task. For sentiment analysis, a task that previously showed a sizable decrease in quality when pruned but displayed only minor quality setbacks when quantized, the results from applying both DQ and IMP (shown in Table 5.4) show a degradation of 5.88% on accuracy and 5.92% on F1 score compared to the baseline, with scores of 0.883 and 0.881 respectively; while still a negative outcome, the accuracy value is better than the one obtained by the solely pruned model, and the F1 score only slightly worsened compared to the pruned value. On the task of named entity recognition, one that showed considerable quality deterioration from both IMP and DQ, the mixture of both (presented in Table 5.5) resulted in far worse results than either of the techniques on their own, with an accuracy of 0.956

(3.19% worse than the baseline) and a F1 score of 0.775 (17.66% worse than the baseline). However, for the models fine-tuned in the task of sentence generation, quantization seems to have mostly nullified the major improvements to the model quality provided by IMP: the quantized and pruned model obtained a BLEU score of 2.373 and a TER score of 1.009 – 0.50% better and 0.68% lower than the baseline results, respectively – showing that, while not as significant as the boost in quality from solely pruning the model, quantizing the pruned model still outperformed the baseline results, and managed to improve the TER score of the model compared to the pruned score.

With a minor overall improvement to the pruned sentiment analysis task, a major decline in quality in the pruned named entity recognition task, and a lower-than-expected degradation on the sentence generation task, we conclude that, while the performance of the model on CPU is always improved, quantizing an already compressed model has significant but inconsistent effects on the quality of the model, and must ultimately be judged on a case-by-case study across tasks to verify if the quality does not severely degrade.

5.5 Environmental Footprint

To properly compare the footprint of our models, we took the average power spent by the GPU to train each model in a specific task, and calculated the estimated CO_2 the models would emit were they to be trained in a data center, as shown in Section 3.4. It is important to note that the high CO_2 emissions observed by the authors in the paper [10] are due to their models being trained using several distributed GPUs; we will instead be comparing the compressed models against our fine-tuned baselines.

Unlike the other evaluation metrics, we calculated the CO_2 emissions considering the total energy cost of training each model, including every model run performed for all learning rates; in the case of IMP, this cost also accounts for the pre-training necessary to prune models. This was done to further resemble a real training procedure for a language model, which usually includes several runs worth of hyperparameter search.

Overall, different model compression techniques have different effects on the CO_2 emissions across fine-tuning each model. Observing the results shown in Table 5.8, we can see that quantization is the best compression technique in terms of energy expenditure due to DQ being applied post-training, requiring no extra model training time. Knowledge distillation proved to be very effective in reducing CO_2 emissions, with a 48.7% reduction over the baseline in the task of sentiment analysis and a 18.2% reduction on the named entity recognition task; however, as previously mentioned in Section 4.3, the workaround used to apply PKD to *GPT-2* based models meant that the teacher model was executed at the same time as the student model, resulting in an increase of CO_2 emissions by 54.6%.

Regarding the pruning compression technique, the estimated CO_2 emissions produced by fine-tuning the models were very similar to the baseline values. However, when adding the cost of pruning the base models using IMP, the resulting emissions became much worse, with large increases of 223% and 342% over the sentiment analysis and named entity recognition task, and a smaller increase of 4% over the sentence generation. This is due to the IMP process requiring extra training time to prune the models,

IMP CO_2 Emissions	All trained models	Best model
BERT	4.250	1.413
GPT-2	3.267	0.620

Table 5.7: Estimated CO_2 emissions (in kg) for all models trained during IMP.

	Baseline	PKD	IMP		DQ‡
			Fine-tune	Full†	
Sentiment Analysis	0.633	0.325	0.633	2.046	0
Named Entity Recognition	0.412	0.337	0.411	1.824	0
Sentence Generation	7.883	12.186	7.591	8.211	0

†Values include CO_2 emissions from previous model training during IMP

‡DQ is a post-training quantization method, and therefore has no additional power usage

Table 5.8: Comparison between estimated CO_2 emissions (in kg) for baseline and compressed models (singular compression techniques only).

which translates to a greater amount of energy spent; in the case of *BERT* based models, the increase is much higher due to the time taken to apply IMP to 70% sparsity compared to the time taken to fine-tune the models, while *GPT-2* based models take relatively less time to prune up to 30% sparsity.

It is worth noting that, taking in consideration the estimated CO_2 emissions for testing all the different training step variations in applying IMP (shown in Table 5.7), this compression technique ends up becoming very influential on the ecological footprint of the model, especially when considering that the emissions would be even greater in a real training procedure where several more runs would be made to cover a wider hyperparameter search. We conclude that, energy-wise, pruning is only a good model compression option when the target model does not require a high level of sparsity to improve its performance, such as the case of *GPT-2* based models.

5.6 Summary

Knowledge distillation proved to be the best compression technique for *BERT* based models, especially for the named entity recognition task, by creating a smaller and faster model while only slightly lowering model quality; however, it proved to be very detrimental to *GPT-2* based models used for the sentence generation task.

Pruning showed no major improvements in the inference time of the BERT based models, obtained worse results than the baseline and provided no size reduction due to the inability to store and load sparse models. However, for the *GPT-2* based models, pruning was the best compression technique, offering a slight reduction in training and inference time while significantly increasing the model quality.

Quantization only slightly lowered the model quality for *BERT* based models, especially for the sentiment analysis task; it was also effective in reducing model size, and showed faster inference times than the baseline when ran on a *Raspberry Pi 4*, a low-end CPU-optimized hardware; however, the

inference speedup times are only worthwhile if the model is to be deployed on such a hardware, since the inference time becomes much slower compared to the baseline when ran on a GPU environment. It severely worsened the model quality for the sentence generation task, which can be attributed to the on-the-fly architectural change required to quantize *GPT-2* based models on PyTorch. Furthermore, the application of quantization on a pruned model proved to be beneficial on some tasks, such as sentence generation (which had outstanding results when pruned but very low results when only quantized) or sentiment analysis (which had low scores for pruned models but good results for quantized models), but showed no major improvements compared to any of the individual compression methods, leading to the conclusion that quantization can be applied on top of other compression techniques to take advantage of the benefits of quantized models, but the results show that it should only be done on a case-by-case basis (or if the quantized benefits outweigh the overall score reduction).

Regarding the environmental footprint, compression techniques that use no extra training time (such as dynamic quantization) or that generate smaller models which require less time to train (such as knowledge distillation) are preferable, since training time is directly linked to the estimated CO_2 emissions of a model. Pruning proved to be the worst performing compression technique due to the large amount of pre-training required to sparsify the model before fine-tuning; additionally, the workaround required to apply knowledge distillation to the sentence generation task meant that the model training was much longer, resulting in higher power expenditure that lead to higher estimated CO_2 emissions.

A summarized version of the comparisons between the resulting metrics can be seen in the Appendix Table A.1.

Chapter 6

Conclusions and Future Work

This work contributed to the study of model compression in NLP, by experimenting several common model compression techniques and drawing conclusions in regards to the trade-off between performance and computational resource usage. We present our achievements in Section 6.1 and possible directions regarding future work in Section 6.2.

While large NLP models keep obtaining state-of-the-art results, we believe model compression can compete with these results while simultaneously making the most out of the model at hand. Further exploring model compression solutions can bring major advantages to the deployment of NLP models, not only for low-end hardware and mobile devices which are present all around us nowadays, but also for environments that require low latency usage or restricted resource expenditure. We also believe model compression solutions are a viable solution for the ever-increasing problem of large models requiring large amounts of energy consumption, the huge monetary costs attached to it, and the worrisome consequence of the carbon footprint produced by it.

6.1 Achievements

When we began this work, we sought to evaluate several common model compression techniques in order to pick the one with best results, keeping in mind the ease of use and trade-offs between performance and computational resource usage. We found that different model compression techniques have different effects based on the architecture of the target model, as well as the NLP task it is fine-tuned on.

Regarding the trade-off in model performance and computational resources required, we conclude that knowledge distillation is the best compression choice for *BERT* based models such as the ones used for sentiment analysis and named entity recognition, having a very low reduction in model evaluation metrics while achieving a major speed-up in inference time and good model size reduction. For *GPT-2* based models, such as the one used for sentence generation, we conclude that pruning has the better effect on the quality of the model, with sizable improvement over the evaluation metrics and a faster inference time, though the effective size of the model remains the same as the original uncom-

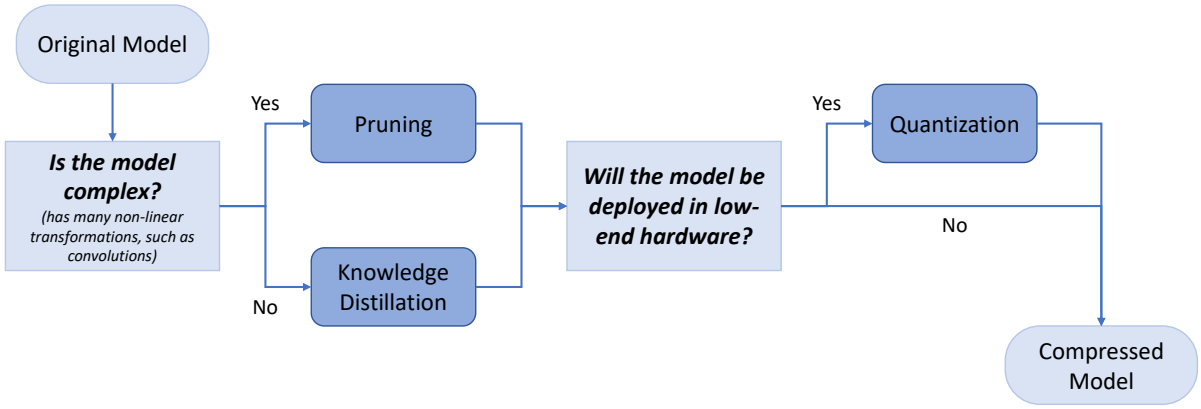


Figure 6.1: A simplified flowchart guide on how to best compress a NLP model, based on our findings.

pressed model.

We were unable to study the effects of quantization-aware training on language models. Nevertheless, we conclude that post-training quantization is ideal for models meant to be deployed in low-end hardware with no GPU; while overall degrading the quality of the model, quantization is effective at compressing the size of a model down to a fraction of the size of the original, and can be applied after other compression techniques to make use of the improvements these may bring to the model, with the downside of lowering or even negating these improvements.

Furthermore, we conclude that the usage of model compression techniques that require extra training besides fine-tuning are noticeably detrimental to the ecological footprint of the model, and compression techniques that reduce the complexity and size of the model before training (or that require no additional training, in the case of post-training compression) should always be preferred.

Based on our results, we outlined a general flowchart (displayed in Figure 6.1) that can be followed to effectively compress a language model while maintaining a good balance between the model quality and performance, and the computational resources needed to store, run and further train the model. We want to stress that this flowchart is made from the preliminary conclusions we drew from our study, and in order to accurately streamline what compression techniques should be picked for models in general, more research and experiments would have to be done in regards to the model compression techniques applied, the language models and NLP tasks used, the training and testing environments, among others; nevertheless, we believe this flowchart to be a broad enough guide to follow. Lastly, we want to underline that, ultimately, the best way to compress any given language model is to test several compression techniques and combinations thereof, since results vary from model to model and task to task.

6.2 Future Work

Although this work contributed to the study of the impact of model compression techniques in NLP, there are several important paths along this line of work that would benefit from better research. For

knowledge distillation, an avenue of research worth broadening would be the study of different student model architectures, not only comparing the resulting model quality but also its versatility regarding different teacher model architectures and complexities, different NLP tasks, and different distillation techniques. Regarding quantization, there is some work to be done on evaluating and comparing different quantization schemes against different NLP tasks and when applied in conjunction with other compression techniques; such schemes can range from less-compressed alternatives such as 16-bit quantization, to more extreme quantization solutions such as GOBO [40], Q-BERT [41], or even BinaryBERT [42], which quantize models to 3-bit, 2-bit and 1-bit – weight binarization – respectively. We briefly covered the combination of different compression techniques in this work, which leads into another interesting study direction to pursue; although we believe not all possible permutations between model compression techniques are fruitful, there is some interest in evaluating the best combinations given the varied results obtained from our experiments.

On a final note, we would like to bring more focus to the usefulness of model compression as the complexity of NLP models keeps increasing; it may even be reaching cost-related roadblocks, in which case compression becomes even more important. There are still several compression features missing, undeveloped or roughly integrated into popular neural network frameworks like PyTorch due to their categorization as experimental features, even though there are many years of work related to this field of study. Some missing features include the saving and loading of sparse and quantized models, or the full implementation of well-developed and tested compression techniques such as quantization-aware training. With these features in place, the study and application of model compression could be further streamlined, resulting in wider appeal and usage across deployed language models as an extra step in fine-tuning.

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All you Need. *Advances in Neural Information Processing Systems*, 30:5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language Models are Unsupervised Multitask Learners, 2019. URL https://www.semanticscholar.org/paper/Language-Models-are-Unsupervised-Multitask-Learners-Radford-Wu/9405cc0d6169988371b2755e573cc28650d14dfe.

[4] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv:1909.08053 [cs]*, Mar. 2020. URL http://arxiv.org/abs/1909.08053. arXiv: 1909.08053.

[5] Microsoft. Turing-NLG: A 17-billion-parameter language model by Microsoft, Feb. 2020. URL https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/.

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*, July 2020. URL http://arxiv.org/abs/2005.14165. arXiv: 2005.14165.

[7] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster,

cheaper and lighter. *arXiv:1910.01108 [cs]*, Feb. 2020. URL http://arxiv.org/abs/1910.01108. arXiv: 1910.01108.

[8] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. Hechtman. Mesh-TensorFlow: Deep Learning for Supercomputers. In *Advances in Neural Information Processing Systems*, volume 31, pages 10414–10423. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/3a37abdeefe1dab1b30f7c5c7e581b93-Paper.pdf.

[9] C. Kaiser. How much difference do GPUs make in model serving?, Jan. 2020. URL https://medium.com/@calebkaiser/how-much-difference-do-gpus-make-in-model-serving-c40b885ac096.

[10] E. Strubell, A. Ganesh, and A. McCallum. Energy and Policy Considerations for Deep Learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1355. URL https://www.aclweb.org/anthology/P19-1355.

[11] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso. The Computational Limits of Deep Learning. *arXiv:2007.05558 [cs, stat]*, July 2020. URL http://arxiv.org/abs/2007.05558. arXiv: 2007.05558.

[12] O. Sharir, B. Peleg, and Y. Shoham. The Cost of Training NLP Models: A Concise Overview. *arXiv:2004.08900 [cs]*, Apr. 2020. URL http://arxiv.org/abs/2004.08900. arXiv: 2004.08900.

[13] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A Survey of Model Compression and Acceleration for Deep Neural Networks. *arXiv:1710.09282 [cs]*, June 2020. URL http://arxiv.org/abs/1710.09282. arXiv: 1710.09282.

[14] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 535–541, Philadelphia, PA, USA, Aug. 2006. Association for Computing Machinery. ISBN 9781595933393. doi: 10.1145/1150402.1150464. URL https://doi.org/10.1145/1150402.1150464.

[15] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531 [cs, stat]*, Mar. 2015. URL http://arxiv.org/abs/1503.02531. arXiv: 1503.02531.

[16] S. Sun, Y. Cheng, Z. Gan, and J. Liu. Patient Knowledge Distillation for BERT Model Compression. *arXiv:1908.09355 [cs]*, Aug. 2019. URL http://arxiv.org/abs/1908.09355. arXiv: 1908.09355.

[17] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova. Well-Read Students Learn Better: On the Importance of Pre-training Compact Models. *arXiv:1908.08962 [cs]*, Sept. 2019. URL http://arxiv.org/abs/1908.08962. arXiv: 1908.08962.

[18] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv:1710.01878 [cs, stat]*, Nov. 2017. URL http://arxiv.org/abs/1710.01878. arXiv: 1710.01878.

[19] Y. L. Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems 2*, pages 598–605. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, June 1990. ISBN 9781558601000.

[20] B. Hassibi, D. G. Stork, G. Wolff, and T. Watanabe. Optimal brain surgeon: extensions and performance comparisons. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 263–270, Denver, Colorado, Nov. 1993. Morgan Kaufmann Publishers Inc.

[21] M. A. Carreira-Perpinan and Y. Idelbayev. "Learning-Compression" Algorithms for Neural Net Pruning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8532–8541, June 2018. doi: 10.1109/CVPR.2018.00890. ISSN: 2575-7075.

[22] M. A. Gordon, K. Duh, and N. Andrews. Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning. *arXiv:2002.08307 [cs]*, May 2020. URL http://arxiv.org/abs/2002.08307. arXiv: 2002.08307.

[23] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, June 2018. doi: 10.1109/CVPR.2018.00286. ISSN: 2575-7075.

[24] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv:1412.6115 [cs]*, Dec. 2014. URL http://arxiv.org/abs/1412.6115. arXiv: 1412.6115.

[25] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *arXiv:2102.00554 [cs]*, Jan. 2021. URL http://arxiv.org/abs/2102.00554. arXiv: 2102.00554.

[26] J. Dodge, S. Gururangan, D. Card, R. Schwartz, and N. A. Smith. Show Your Work: Improved Reporting of Experimental Results. *arXiv:1909.03004 [cs, stat]*, Sept. 2019. URL http://arxiv.org/abs/1909.03004. arXiv: 1909.03004.

[27] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat. Q8BERT: Quantized 8Bit BERT. *arXiv:1910.06188 [cs]*, Oct. 2019. URL http://arxiv.org/abs/1910.06188. arXiv: 1910.06188.

[28] T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, Z. Wang, and M. Carbin. The Lottery Ticket Hypothesis for Pre-trained BERT Networks. *arXiv:2007.12223 [cs, stat]*, Oct. 2020. URL http://arxiv.org/abs/2007.12223. arXiv: 2007.12223.

[29] J. Frankle and M. Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. *arXiv:1803.03635 [cs]*, Mar. 2019. URL http://arxiv.org/abs/1803.03635. arXiv: 1803.03635.

[30] S. Prasanna, A. Rogers, and A. Rumshisky. When BERT Plays the Lottery, All Tickets Are Winning. *arXiv:2005.00561 [cs]*, Oct. 2020. URL http://arxiv.org/abs/2005.00561. arXiv: 2005.00561.

[31] EEA. Greenhouse gas emission intensity of electricity generation — European Environment Agency. URL https://www.eea.europa.eu/data-and-maps/daviz/co2-emission-intensity-6.

[32] O. US EPA. Emissions & Generation Resource Integrated Database (eGRID), July 2020. URL https://www.epa.gov/egrid.

[33] F. Wu. Webinar: Uptime Institute Global Data Center Survey 2020. URL https://uptimeinstitute.com/webinars/webinar-uptime-institute-global-data-center-survey-2020.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[35] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

[36] E. F. Tjong Kim Sang and F. De Meulder. Introduction to the CoNLL-2003 shared task: language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 142–147, Edmonton, Canada, May 2003. Association for Computational Linguistics. doi: 10.3115/1119176.1119195. URL https://doi.org/10.3115/1119176.1119195.

[37] S. Zhang, E. Dinan, J. Urbanek, A. Szlam, D. Kiela, and J. Weston. Personalizing Dialogue Agents: I have a dog, do you have pets too? *arXiv:1801.07243 [cs]*, Sept. 2018. URL http://arxiv.org/abs/1801.07243. arXiv: 1801.07243.

[38] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer Sentinel Mixture Models. *arXiv:1609.07843 [cs]*, Sept. 2016. URL http://arxiv.org/abs/1609.07843. arXiv: 1609.07843.

[39] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books. *arXiv:1506.06724 [cs]*, June 2015. URL http://arxiv.org/abs/1506.06724. arXiv: 1506.06724.

[40] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos. GOBO: Quantizing Attention-Based NLP Models for Low Latency and Energy Efficient Inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 811–824, Oct. 2020. doi: 10.1109/MICRO50266.2020.00071.

[41] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. *arXiv:1909.05840 [cs]*, Sept. 2019. URL `http://arxiv.org/abs/1909.05840`. arXiv: 1909.05840.

[42] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King. BinaryBERT: Pushing the Limit of BERT Quantization. *arXiv:2012.15701 [cs]*, July 2021. URL `http://arxiv.org/abs/2012.15701`. arXiv: 2012.15701.

Appendix A

Compression Results

This appendix contains the results of the remaining model metrics we evaluated for every model, as well as a result summary for all of the metrics regarding compressed models.

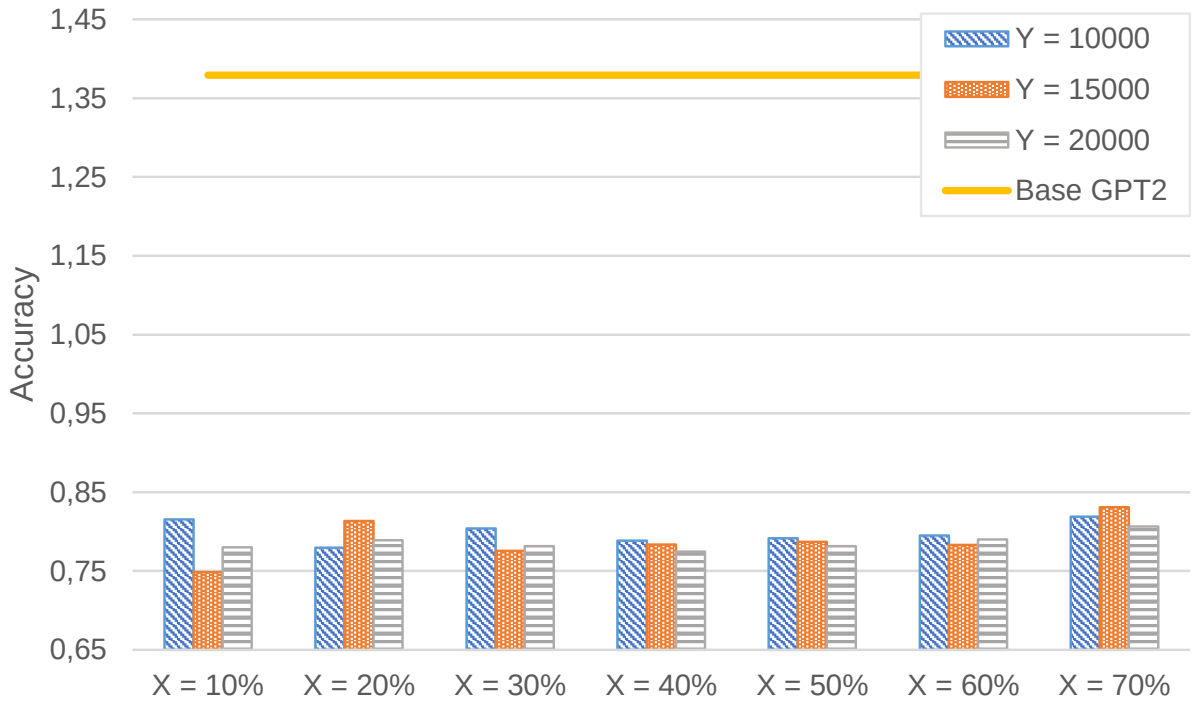
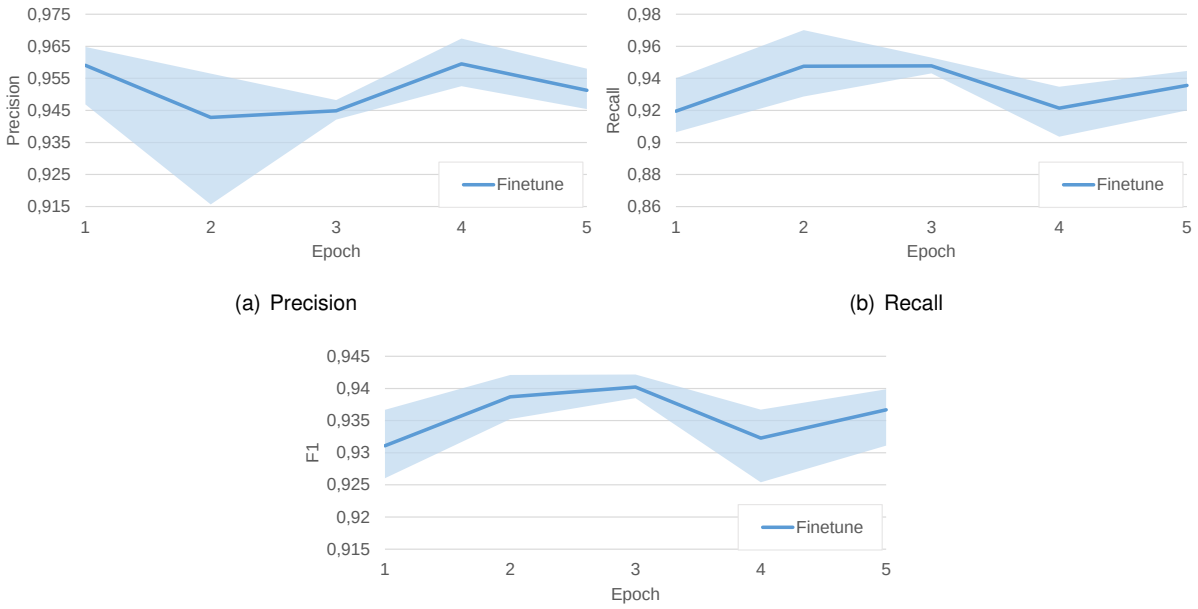


Figure A.1: Accuracy metric for *GPT-2* models after applying iterative magnitude pruning to X% sparsity, by pruning 10% of the weights every Y training steps. The models were trained on the *WikiText-103* dataset. The accuracy of the base *GPT-2* model is displayed for comparison.

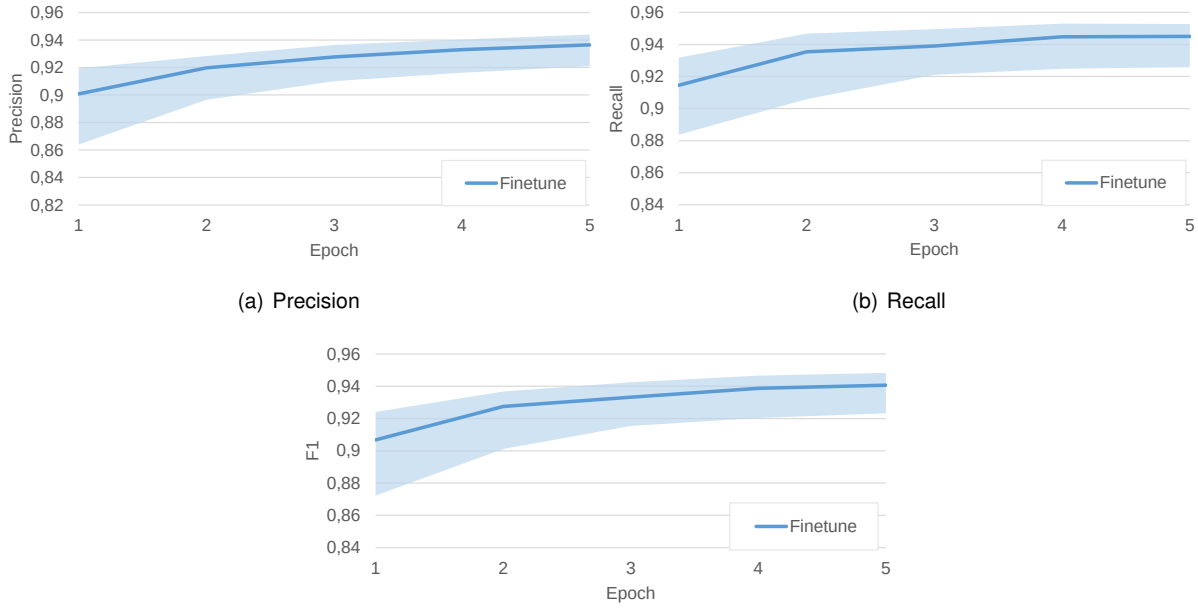


(a) Precision

(b) Recall

(c) F1

Figure A.2: Additional fine-tuned results for the sentiment analysis task. Complementary to Figure 4.1.

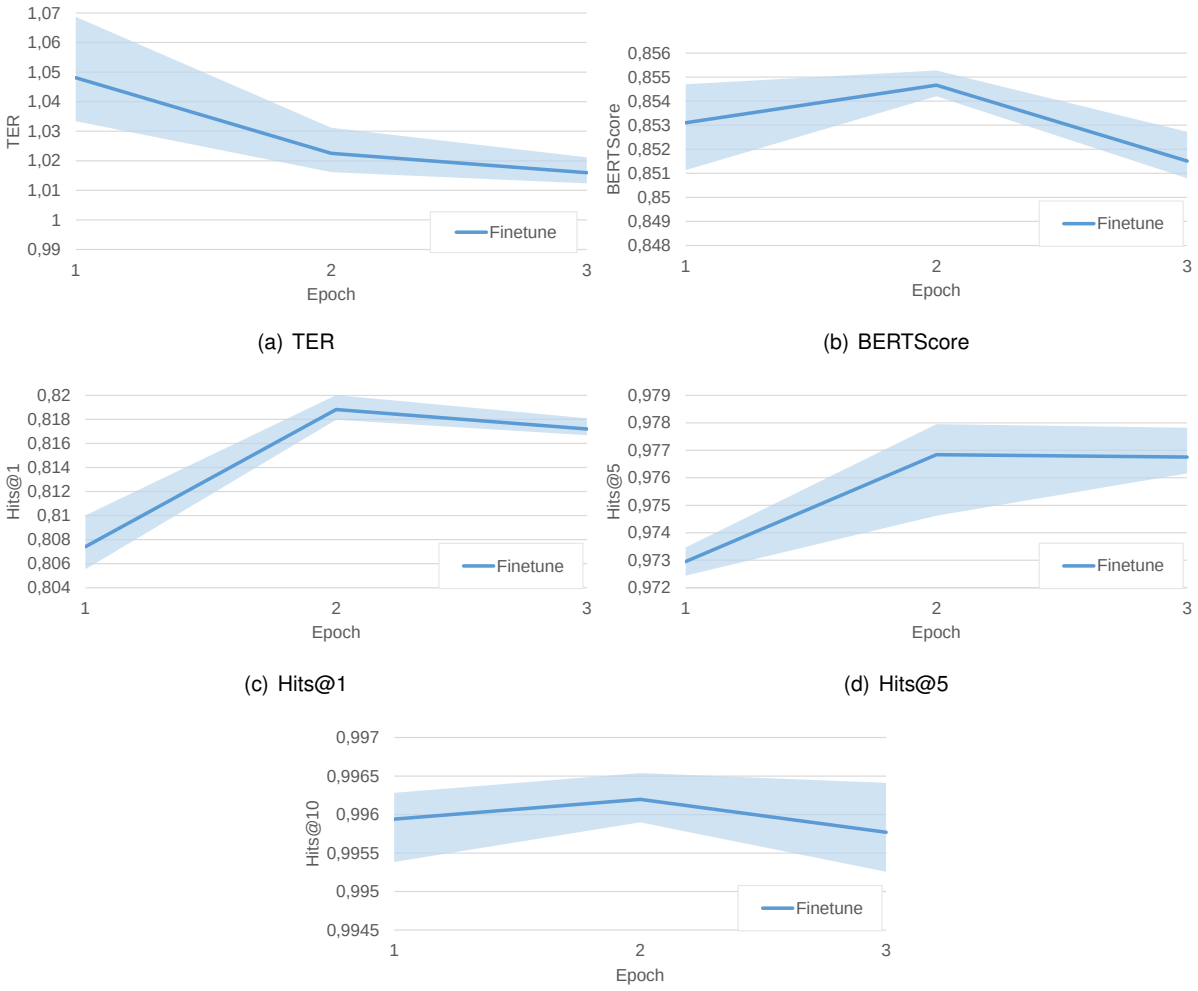


(a) Precision

(b) Recall

(c) F1

Figure A.3: Additional fine-tuned results for the named entity recognition task. Complementary to Figure 4.1.



(a) TER

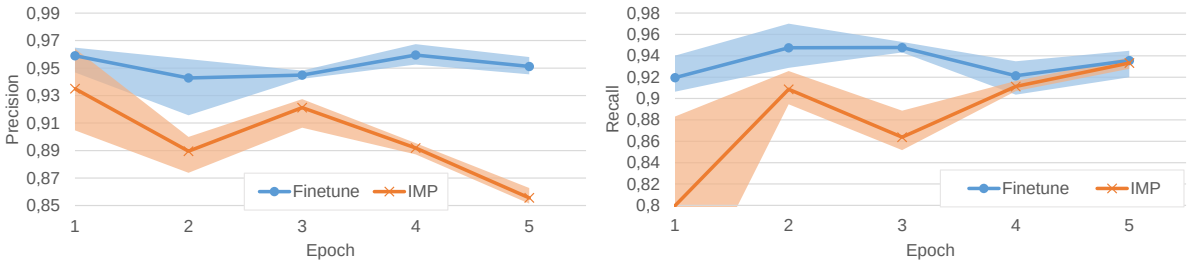
(b) BERTScore

(c) Hits@1

(d) Hits@5

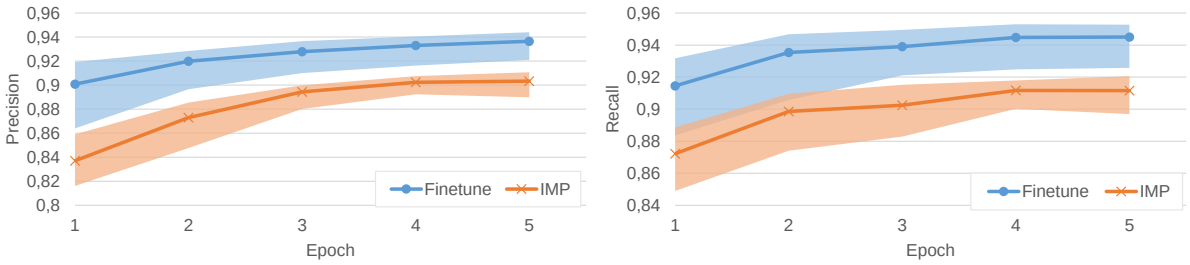
(e) Hits@10

Figure A.4: Additional fine-tuned results for the sentence generation task. Complementary to Figure 4.1.



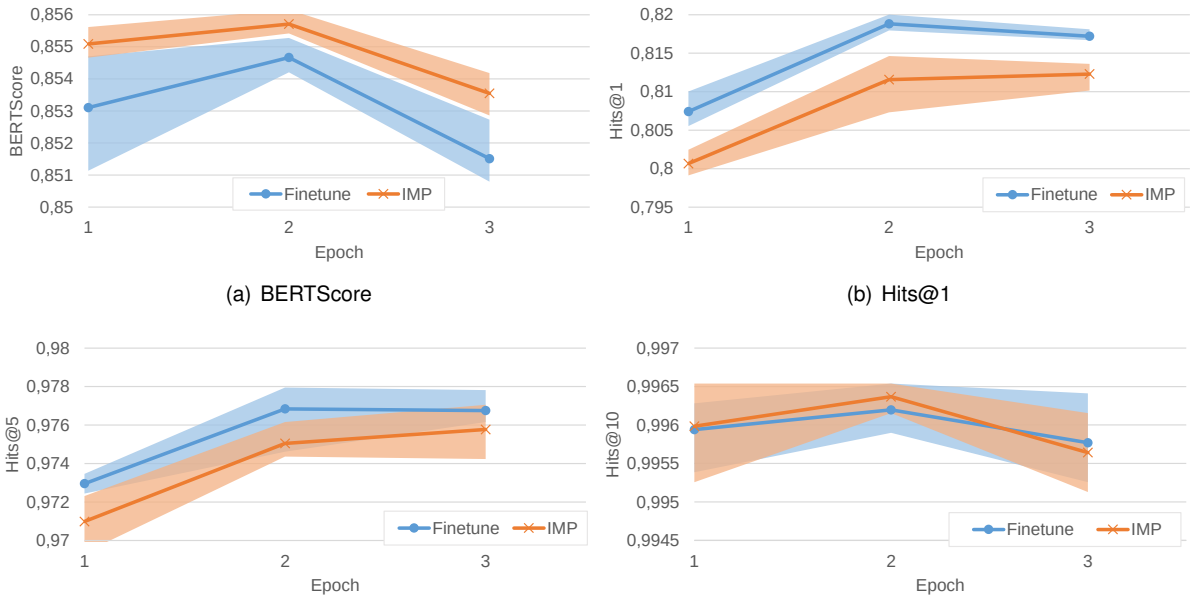
(a) Precision (b) Recall

Figure A.5: Additional experiment results for pruned models fine-tuned in sentiment analysis. Complementary to Figure 5.4.



(a) Precision (b) Recall

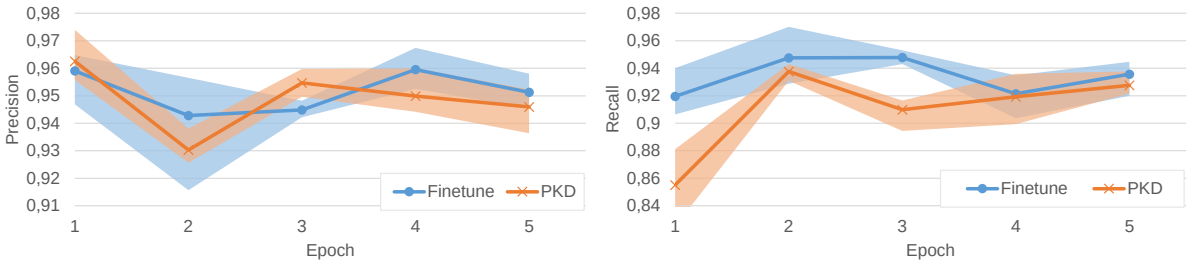
Figure A.6: Additional experiment results for pruned models fine-tuned in named entity recognition. Complementary to Figure 5.5.



(a) BERTScore (b) Hits@1

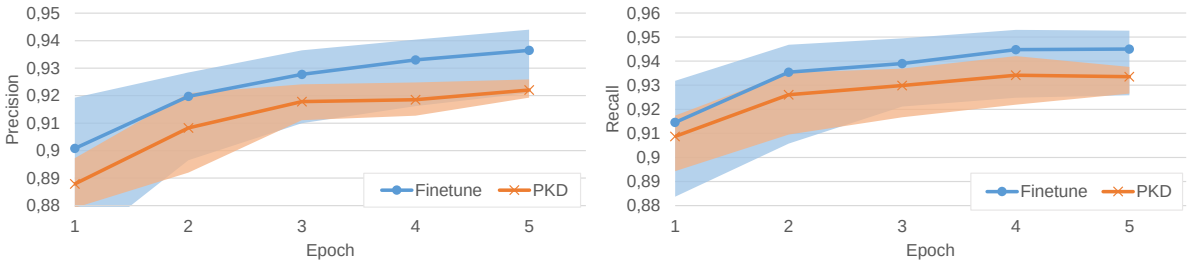
(c) Hits@5 (d) Hits@10

Figure A.7: Additional experiment results for pruned models fine-tuned in sentence generation. Complementary to Figure 5.6.



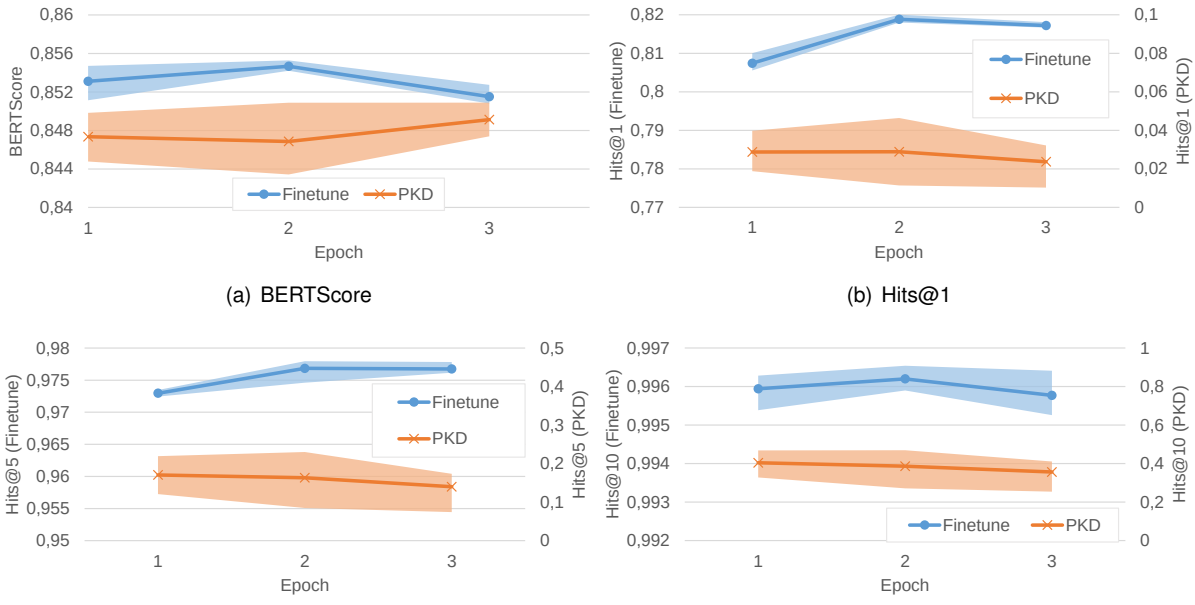
(a) Precision (b) Recall

Figure A.8: Additional experiment results for distilled models fine-tuned in sentiment analysis. Complementary to Figure 5.1.



(a) Precision (b) Recall

Figure A.9: Additional experiment results for distilled models fine-tuned in named entity recognition. Complementary to Figure 5.2.



(a) BERTScore (b) Hits@1

(c) Hits@5 (d) Hits@10

Figure A.10: Additional experiment results for distilled models fine-tuned in sentence generation. Complementary to Figure 5.3. Note that, due to the large difference between results, the Hits@1/5/10 metrics are displayed with two separate value axis.

Task	Metrics	Baseline	Knowledge Distillation	Pruning	Quantization	Quantization + Pruning
Sentiment Analysis	Accuracy	0.938	0.931 (↓ 0.76%)	0.874 (↓ 6.90%)	0.934 (↓ 0.51%)	0.883 (↓ 5.88%)
	F1	0.937	0.929 (↓ 0.82%)	0.883 (↓ 5.71%)	0.927 (↓ 1.00%)	0.881 (↓ 5.92%)
	Precision	0.951	0.946 (↓ 0.56%)	0.856 (↓ 10.06%)	**0.965 (↑ 1.40%)**	0.900 (↓ 5.38%)
	Recall	0.936	0.928 (↓ 0.85%)	0.933 (↓ 0.25%)	0.908 (↓ 2.94%)	0.884 (↓ 5.51%)
Named Entity Recognition	Accuracy	0.988	0.985 (↓ 0.27%)	0.981 (↓ 0.68%)	0.979 (↓ 0.91%)	0.956 (↓ 3.19%)
	F1	0.941	0.928 (↓ 1.38%)	0.907 (↓ 3.53%)	0.894 (↓ 5.00%)	0.775 (↓ 17.66%)
	Precision	0.936	0.922 (↓ 1.54%)	0.903 (↓ 3.54%)	0.890 (↓ 4.99%)	0.798 (↓ 14.78%)
	Recall	0.945	0.934 (↓ 1.21%)	0.912 (↓ 3.53%)	0.898 (↓ 4.96%)	0.768 (↓ 18.72%)
Sentence Generation	BLEU	2.362	1.975 (↓ 16.37%)	**2.682 (↑ 13.56%)**	2.076 (↓ 12.10%)	**2.373 (↑ 0.50%)**
	TER	1.016	1.035 (↑ 1.86%)	1.024 (↑ 0.78%)	**1.016 (↓ 0.01%)**	**1.009 (↓ 0.68%)**
	BERTScore	0.852	0.849 (↓ 0.28%)	**0.854 (↑ 0.24%)**	0.849 (↓ 0.24%)	**0.852 (↑ 0.00%)**
	Hits@1	0.817	0.024 (↓ 97.10%)	0.812 (↓ 0.60%)	0.809 (↓ 0.96%)	0.803 (↓ 1.72%)
	Hits@5	0.977	0.140 (↓ 85.67%)	0.976 (↓ 0.10%)	0.974 (↓ 0.30%)	0.973 (↓ 0.39%)
	Hits@10	0.996	0.356 (↓ 64.24%)	0.996 (↓ 0.01%)	0.995 (↓ 0.10%)	0.995 (↓ 0.07%)

Table A.1: Comparison between the results of all model compression techniques, and the variance percentage compared to the baseline results. The arrows represent whether the variance is positive or negative. **Filled cells** signify the best scores between the compression techniques, and **bold cells** signify that the score obtained for that compression technique is better than or equal to the baseline score.