# A Study on the Best Way to Compress Natural Language Processing Models

João Antunes, Miguel L. Pardal, Luisa Coheur
*INESC-ID, Instituto Superior Técnico, Universidade de Lisboa*, Lisbon, Portugal
{joao.carlos.antunes, miguel.pardal, luisa.coheur}@tecnico.ulisboa.pt

***Abstract*—Current research in Natural Language Processing shows a growing number of models extensively trained with large computational budgets. However, these models present computationally demanding requirements, preventing them from being deployed in devices with strict resource and response latency limitations. In this paper, we apply state-of-the-art model compression techniques to create compact versions of several of these models. In order to evaluate whether the trade-off between model performance and budget is worthwhile, we evaluate them in terms of efficiency, model simplicity and environmental footprint. We also present a brief comparison between uncompressed and compressed models when running in low-end hardware.**

***Index Terms*—model compression, model evaluation, environmental footprint**

## I. Introduction

Current research and developments in neural network technology has brought a revolution to the Natural Language Processing (NLP) field, favoring bulky models, containing a large number of parameters – model weights and coefficients that change and learn with the training data. Examples of such models are BERT [1] (up to 340 million parameters), Turing-NLG[1] (17 billion parameters) and GPT-3 [2] (a staggering 175 billion parameters), among many others. However, there are several issues with such high-performing language models, considering their demanding requirements. If *hardware* is taken into consideration, fitting a model with a billion parameters in a single advanced data-center GPU is impossible. Even for high-end hardware. Model parallelism circumvents this issue by partitioning large models over several GPUs. 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 (usually for portability, cost, and power consumption reasons) such as smartphones. *Time* is another issue: the latency of the model (to infer a result) is heavily dependent on the environment where the model is deployed. Benchmarking tests[2] have shown that the usage of a GPU provides substantially lower latency than the usage of a CPU for model inference. For devices with hardware constraints, such as smartphones, this effectively leads to an undesirable trade-off between model accuracy and response latency. Additionally, *monetary cost* and *environmental footprint* are often overlooked yet important issues that should be taken into consideration. A recent review [3] estimates that training a model with 1.5 billion parameters costs around \$80k (in U.S. dollars), scaling up to \$1.6m with proper hyperparameter tuning; these figures act as a paywall for developing new models, as few research labs can afford costs of this magnitude. In addition, we are reaching a point where not even those who can make such high monetary investments are willing to do it: the authors of the *GPT-3* model found a mistake when implementing the system, but decided against fixing it *"due to the cost of training it was not feasible to retrain the model"* [2]. Considering the environmental footprint in the training of such models, a recent study [4] reported that a fully trained *BERT* model emits as much $CO_2$ as a trans-American flight from New York to San Francisco, and future models may bring much more drastic effects to the environment.

*Model compression techniques* [5] 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*. In this paper, we compare these three different compression techniques, applied to three fundamental and widely known tasks in NLP – *sentiment analysis*, *named entity recognition* and *dialog-driven sentence generation* – 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. Furthermore, we test the performance of quantized models in a *Raspberry Pi*, an inexpensive computer.

Also, 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 report their best accuracy values, but omit 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. Therefore, in this paper we also propose a well-detailed evaluation of every model: we display the *expected validation performance* [6] for all evaluation metrics, the final size of the model, average training and inference time, computing infrastructure used,

[1]https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/
[2]https://www.joyk.com/dig/detail/1576888233476928

model hyperparameters and dataset splits. Additionally, we compare the training process in terms of training time and power usage, as well as estimate $CO_2$ emissions for the same model to be trained in a data center, to understand whether the reduction in model size and complexity 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.

Results show that *knowledge distillation* is the best compression choice for 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 a good reduction in model size. For 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 uncompressed model.

## II. Related Work

In this Section, we describe model compression techniques, how they are applied to NLP models, and also works concerned with the energy consumption and carbon emissions.

### A. Compression Techniques

*Knowledge distillation* [7], [8] or *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* (Figure 1). This larger model has higher knowledge capacity and can provide high performance but is also computationally expensive to evaluate.

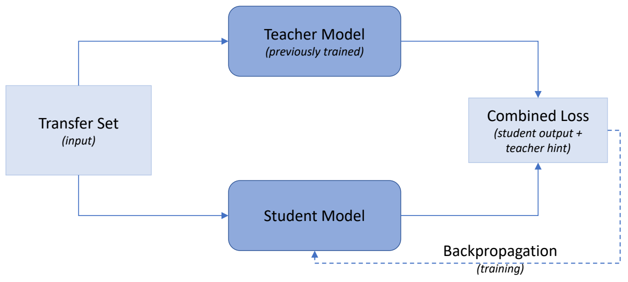


Fig. 1. An overview of teacher-student learning.

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.

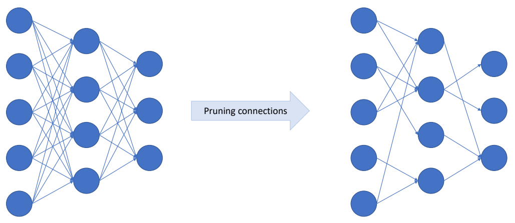


Fig. 2. Weight-based pruning applied to a network.

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 to apply it to pre-trained models instead [9]. 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.

*Pruning* (Figure 2) refers to the removal of redundant and non-informative connections within the network of the model 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 [10]. 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 efficient loading and operations.

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 calculating 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.

More recent approaches [10], [11] 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 [12]. 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.

In what respects *quantization*, a logical step in compressing a model would be to trim down the model parameters themselves. Network *quantization* [13] 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 to the training data. Furthermore, the two techniques are compatible with each other and can be used simultaneously for further model reduction.

Generally, 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.

## B. Compressing NLP models

Model compression has been an especially important field of study for any BERT-based models, which have a large memory footprint and require heavy computing during inference.

Q8BERT [14] is 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. The weight quantization was done during training, using a technique called Quantization-Aware Training (QAT): during model fine-tuning, fake quantization [13] 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 (DQ) – testing was done on several NLP tasks. Results show that QAT outperforms DQ in every task, and even performs better than or equal to the original uncompressed model in certain tasks. However, for the moment, PyTorch does not directly support applying QAT to models that require embedding layers. 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.

DistilBERT [15] was developed by *teaching a smaller version of BERT*, reducing the amount of layers from the regular 12 to just 6, 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. 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. 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, but 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.

In what concerns *pruning*, the computer vision community explored the Lottery Ticket Hypothesis [16], 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, several concurrent studies [12], [17], [18] 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 performance;
- 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, 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 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.

## C. 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 [19]. 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 [20], or $430g$ per kilowatt-hour. Considering that data centers spend a substantial amount of energy maintaining servers 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.

## III. Experiments

In this section, we describe our approach to evaluate the effects of model compression applied to NLP tasks, namely: *sentiment analysis*, *named entity recognition* and *sentence generation*.

### A. Datasets

To train the language models fine-tuned on the sentiment analysis task, we used the *IMDB review* dataset [21] – a group of positive and negative *IMDB* movie reviews. For the named entity recognition task, we made use of the *CoNLL-2003* dataset [22] – a collection of phrases where every word has a corresponding part-of-speech tag, syntactic tag and named entity tag. To train and fine-tune the GPT-2 based conversational model on the task of sentence generation, we used the *Persona-Chat* dataset [23], which is a crowd-sourced collection of over 160 thousand utterances between pairs of personas. Additionally, to prune both BERT and GPT-2 based models, we needed 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 [24], a bundle of several million tokens extracted from verified articles on *Wikipedia*.

### B. Evaluation

Following a set of guidelines for best practices [6], for every model trained and evaluated, we reported a description of the computing infrastructure used during training, the average

|  | Sentiment Analysis | Named Entity Recognition | Sentence Generation |
|---|---|---|---|
| GPU | 1080 Ti (11GB) | TITAN X (12GB) | 1080 Ti (11GB) |
| Data Splits (tr./dev/test) | 50/25/25 | 70/15/15 | 99/0.5/0.5 |
| Epochs | 5 | 5 | 3 |
| Batch Size (train/test) | 8/8 | 32/8 | 2/1 |
| Learning Rate | 3e-5, 1e-4, 3e-4, 5e-5 |  | 6.25e-5, 5e-5, 1e-4 |

TABLE I
General training setup for every task, including hardware and hyperparameter configurations. All settings remained the same across every model, whether compressed or not.

runtime for each approach, the details of dataset splits, the corresponding validation performance for each reported test result, a link to implemented code, the response time of the model during execution, and the size of the model. Additionally, we used heuristics related to energy consumption and environmental footprint [4] 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. 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.

### C. General Training Setup

Across every model trained, we used the PyTorch framework [25]. For the models themselves, we picked *BERT*$_{BASE}$ [1] as the pre-trained base model to accomplish the sentiment analysis[3] and named entity recognition[4] NLP tasks, and *GPT-2*$_{Small}$ [26] for conversation-driven sentence generation[5].

To obtain the baseline from which we can compare the compressed models, we fine-tuned the pre-trained model for the task at hand.

### D. General Testing Setup

We set up a controlled testing environment by maintaining the same testing hardware and hyperparameters across models for the same tasks; all of these testing configurations are presented in Table I. 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; the average GPU power draw for every model trained was queried from the NVIDIA System Management Interface (`nvidia-smi`[6]) in parallel to the model training itself, which gathered and registered the current

[3]https://github.com/Uziskull/lightning-text-classification
[4]https://github.com/Uziskull/BERT-NER
[5]https://github.com/Uziskull/lightning-convai
[6]https://developer.nvidia.com/nvidia-system-management-interface

power draw of the GPU being used, with an interval of 5 seconds.

Every model run was made deterministic by setting a fixed seed for randomness, which prevented variations in weight initialization and data order across runs, thus limiting the variable testing details to just the different learning rates and the different compression techniques applied. This decision 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 themselves.

## *E. Knowledge Distillation*

To distill knowledge, we created 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) act as the teacher model and perform Patient Knowledge Distillation (PKD) [27]. 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 the first six layers of pre-trained weights were used for initialization. To train the student models, we followed the PKD-Skip strategy outlined by the authors.

To transfer the knowledge from the teacher model to the target student model, we took the entire training dataset used for the student model and ran it through the teacher model; the resulting output from the intermediate and final layers was the knowledge set to be distilled to the student model. We were unable to perform the same teacher knowledge gathering for the sentence generation task since the resulting dataset with extra knowledge was too large to be able to be loaded during student training. To get around this issue, we ran the teacher model during the student training to get the necessary resulting values on-the-fly; while the models were successfully distilled, the resulting overloading of the GPU had some drawbacks related to training time and power spent.

## *F. Pruning*

Following the Lottery Ticket Hypothesis, to prune every task-specific model, we first took the pre-trained base model for each task and trained it further while applying Iterative Magnitude Pruning (IMP) [17], which prunes the attention heads of the model over several training epochs. 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. We used WikiText-103 [24] corpus for the required extra training on the pre-trained base models.

To apply IMP, pruned 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. Since the research conducted on the Lottery Ticket Hypothesis was done on BERT based models, the universality of task transferal observed at 70% sparsity could not be directly inferred to the GPT-2 model; as such, we decided to conduct some extra testing to observe what the ideal sparsity should be, by applying IMP to the pre-trained GPT-2 model, 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%).

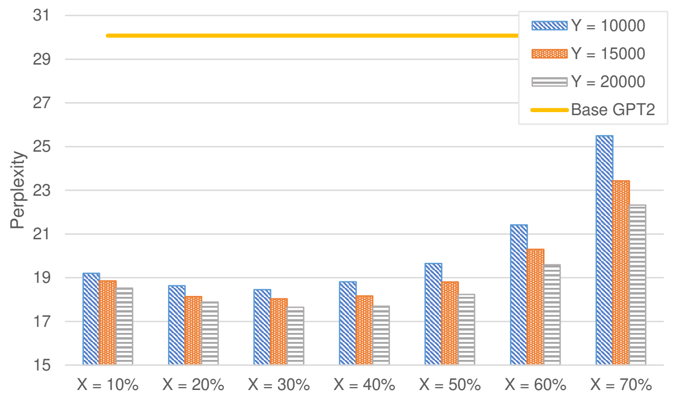


Fig. 3. Experiment results for GPT-2 models after applying IMP.

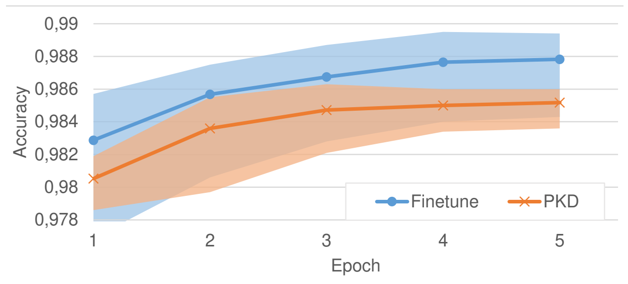
Figure 3 shows the results after applying iterative magnitude pruning to $X\%$ sparsity, by pruning 10% of the weights every $Y$ training steps. The models were trained on the *WikiText-103* dataset. The perplexity of the base GPT-2 model is displayed for comparison. The obtained results show a trend throughout all 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. As such, 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 decided to use the perplexity metric to choose 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.

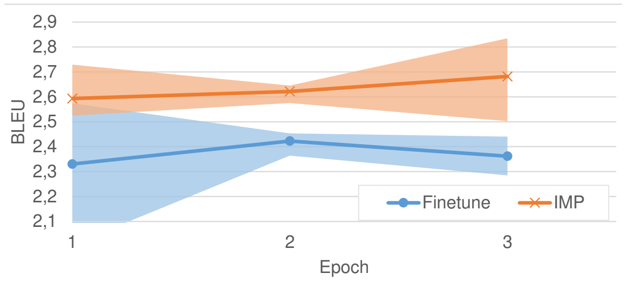
## *G. Quantization*

To quantize every model, we applied DQ to the fine-tuned models used as baseline with no additional training time, since it is a post-training compression technique; the weights of the models were quantized to 8bit (using the *QNNPACK*[7] backend). GPT-2 based models make use of one-dimensional convolutions over incoming data (Conv1D layers), which are not supported by the DQ backend. In order to be able to quantize GPT-2, every Conv1D layer had to be swapped with Linear layers after loading the model; while this change allowed for GPT-2 models to be quantized, it had some negative effects on the performance of those models.

[7]https://github.com/pytorch/QNNPACK



(a) Expected validation accuracy for distilled models (PKD) fine-tuned in named entity recognition



(b) Expected validation BLEU scores for pruned models (IMP) fine-tuned in sentence generation

| | Model Size (MB) | Inf. Time CPU (s) | Inf. Time GPU (s) | Accuracy | F1 |
|---|---|---|---|---|---|
| Baseline | 440 | 10.000 | 0.003 | 0.938 | 0.937 |
| DQ | 175 | 4.850 | 0.286 | 0.934 | 0.927 |

(c) Experiment results for quantized models (DQ) fine-tuned in sentiment analysis

Fig. 4. Excerpt of compressed model results compared against the respective baseline models.

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* (4GB 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.

### *H. Quantization + Pruning*

Taking advantage of the compatibility between both compression techniques, we used *quantization-aware training* on the previously pruned model during fine-tuning of the task. To test the combined effects of quantization and pruning, we applied DQ to the models previously pruned via IMP and evaluated the resulting models. 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.

## IV. Results

In this section, we present the obtained results. Table II shows 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.

### *A. Knowledge Distillation*

For the sentiment analysis and named entity recognition tasks, PKD proved to be a very effective compression technique; none of the recorded metrics for the distilled models degraded more than $1\%$ compared to the fine-tuned baseline. The expected performance ranges of the compressed model and the baseline overlap (as displayed in Figure 4(a)), implying that a well-trained distilled model could outperform the original uncompressed model, although this verification is outside the scope of this work. The small size of the distilled models (270MB – a $39\%$ reduction) resulted in shorter training and inference times, with the sentiment analysis task achieving a $79\%$ latency improvement over the baseline model. However, PKD was detrimental to the sentence generation task, with a severe $16.37\%$ drop in the BLEU score compared to the baseline. We attribute this to the architecture of the student model being too compact to properly learn from the complex *GPT-2* teacher model, and believe that distillation could be viable for this task with a different student model architecture. Additionally, due to the distilling workaround described in Section III-E, the training time of the model was extended by $52\%$, consequently increasing the power spent and estimated $CO_2$ emissions.

### *B. Pruning*

Overall, IMP showed reasonable worsening of model quality for the sentiment analysis and named entity recognition tasks, with no major difference in training or inference time, and consequential power expenditure or $CO_2$ emissions. On the other hand, for sentence generation, the pruned models improved considerably compared to the baseline (shown in Figure 4(b)), with a substantial $13.56\%$ increase of the BLEU score; additionally, the training and inference times for the pruned model are $4\%$ and $9\%$ smaller than the ones reported for the baseline, which we attribute to the more efficient computation of the complex convolution layers present in *GPT-2* models, given that $30\%$ of the weights are set to zero.

Ultimately, no size improvement was obtained since no method of sparse matrix representation would allow for a smaller sized weight storage without additional detrimental computational overhead [28]; additionally, PyTorch currently offers no way of saving or loading sparse weights.

| Task | Metrics | Baseline | PKD | IMP | DQ | DQ + IMP |
|---|---|---|---|---|---|---|
| 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 Recogn. | 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 II
RESULTS OF THE MODEL COMPRESSION TECHNIQUES

## C. Quantization

Unlike the other compression techniques, 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, with the task of sentiment analysis (displayed in Table 4(c)) only lowering its accuracy and F1 scores by $0.51\%$ and $1.00\%$, respectively. For the sentence generation task, DQ showed lowered scores overall, which we attribute in part to the architectural change required for quantizing *GPT-2* based models, previously mentioned in Section III-G. All models showed a reduction in model size, with *BERT* based models being compressed to $60\%$ of the original model size and *GPT-2* based models ending up with a size of 280MB ($57\%$ of the starting size); this reduction is only reflected in the size loaded in memory, however, since there is no current way of saving or directly loading a quantized model in PyTorch.

Regarding the CPU testing done on the *Raspberry Pi*, there was a noticeable improvement on inference times, with $51.50\%$, $59.89\%$ and $63.10\%$ faster performance on the tasks of sentiment analysis, named entity recognition and sentence generation, respectively. However, this inference speedup is only seen within low-end CPU-optimized hardware; GPUs manage to execute integer weight operations faster than the *Raspberry Pi* and obtain lower inference times for both quantized and baseline models, with baseline models being several orders of magnitude faster.

## D. Quantization + Pruning

Quantizing the already pruned models showed 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, all relative to the pruned models themselves. These changes were related to both the performance of the pruned models, as well as the influence of quantization on the models. While the performance of the model on CPU was still improved (much like the former quantization results show),

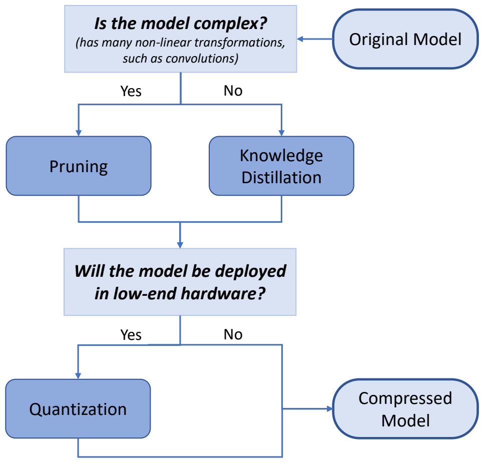

Is the model complex?
(has many non-linear transformations, such as convolutions)
Original Model
Yes
No
Pruning
Knowledge Distillation
Will the model be deployed in low-end hardware?
Yes
No
Quantization
Compressed Model

Fig. 5. A simplified flowchart guide on how to best compress a NLP model.

the effects of quantizing the pruned models were somewhat inconsistent across tasks, further confirming that quantization should only be applied on a case-by-case basis.

## E. Environmental Footprint

Overall, compression techniques that require lower training time spend less energy doing so, resulting in lower estimated $CO_2$ emissions. DQ was applied post-training, thus having a null $CO_2$ emission value. For the compression techniques applied pre-training, PKD was the one with lower energy expenditure and consequential environmental impact, due to the smaller model being able to be trained faster; this was not the case for the sentence generation task which, due to the workaround described in Section III-E, spent the energy required to both train the student model and execute the teacher model. IMP had an ecological footprint similar to the baseline models, even showing smaller energy costs for the sentence generation task, but when accounting for the energy spent pruning the pre-trained models to the required sparsity before fine-tuning, the estimated $CO_2$ emissions become much higher, with increases as high as $342\%$ (for the named entity recognition task).

## V. Conclusion and Future Work

In this paper, we applied several common model compression techniques to NLP tasks and compared the trade-off between performance and computational resource usage. Based on our results, we outlined a flowchart (shown in Figure 5) for effectively compressing a language model; this flowchart is based on preliminary conclusions we drew from our study, and more research and experiments would have to be done in order to generalize this across all language models targeted for NLP usage. 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.

For future work, we believe more extensive research could be done on each of the compression techniques, such as testing several student model architectures for knowledge distillation and experimenting with different compression schemes and bit depths for quantization. We also believe model compression should be fully integrated and properly implemented in popular model frameworks like PyTorch, instead of being available only as experimental features, to promote its usage across future language models and further reduce computational costs for training such complex models.

## Acknowledgements

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT), project UIDB/50021/2020 (INESC-ID multi-annual funding); by FEDER, Programa Operacional Regional de Lisboa, Agência Nacional de Inovação (ANI), and CMU Portugal, under the project Ref. 045909 (MAIA), and under project LISBOA-01-0247-FEDER-045385 (Intelligent Agents Generator).

## References

[1] 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)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.

[2] 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*, Jul. 2020.

[3] O. Sharir, B. Peleg, and Y. Shoham, "The Cost of Training NLP Models: A Concise Overview," *arXiv:2004.08900*, Apr. 2020.

[4] 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*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 3645–3650.

[5] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A Survey of Model Compression and Acceleration for Deep Neural Networks," *arXiv:1710.09282*, Jun. 2020.

[6] J. Dodge, S. Gururangan, D. Card, R. Schwartz, and N. A. Smith, "Show Your Work: Improved Reporting of Experimental Results," *arXiv:1909.03004*, Sep. 2019.

[7] 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*, ser. KDD '06. Philadelphia, PA, USA: Association for Computing Machinery, Aug. 2006, pp. 535–541.

[8] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv:1503.02531*, Mar. 2015.

[9] 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*, Sep. 2019.

[10] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv:1710.01878*, Nov. 2017.

[11] 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*, Jun. 2018, pp. 8532–8541, iSSN: 2575-7075.

[12] M. A. Gordon, K. Duh, and N. Andrews, "Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning," *arXiv:2002.08307*, May 2020.

[13] 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*, Jun. 2018, pp. 2704–2713, iSSN: 2575-7075.

[14] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8BERT: Quantized 8Bit BERT," *arXiv:1910.06188*, Oct. 2019.

[15] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," *arXiv:1910.01108*, Feb. 2020.

[16] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," *arXiv:1803.03635*, Mar. 2019.

[17] 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*, Oct. 2020.

[18] S. Prasanna, A. Rogers, and A. Rumshisky, "When BERT Plays the Lottery, All Tickets Are Winning," *arXiv:2005.00561*, Oct. 2020.

[19] EEA, "Greenhouse gas emission intensity of electricity generation — European Environment Agency."

[20] O. US EPA, "Emissions & Generation Resource Integrated Database (eGRID)," Jul. 2020.

[21] 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*. Portland, Oregon, USA: Association for Computational Linguistics, Jun. 2011, pp. 142–150.

[22] 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*, ser. CONLL '03. Edmonton, Canada: Association for Computational Linguistics, May 2003, pp. 142–147.

[23] 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*, Sep. 2018.

[24] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer Sentinel Mixture Models," *arXiv:1609.07843*, Sep. 2016.

[25] 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 *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[26] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," 2019.

[27] S. Sun, Y. Cheng, Z. Gan, and J. Liu, "Patient Knowledge Distillation for BERT Model Compression," *arXiv:1908.09355*, Aug. 2019.

[28] 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*, Jan. 2021.