UNIVERSIDADE FEDERAL FLUMINENSE

RICARDO DE SOUZA MOURA

LESS IS MORE: PRUNING BERTWEET ARCHITECTURE FOR FINETUNING IN TWITTER SENTIMENT ANALYSIS

NITERÓI

2023

RICARDO DE SOUZA MOURA

LESS IS MORE: PRUNING BERTWEET ARCHITECTURE FOR FINETUNING IN TWITTER SENTIMENT ANALYSIS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação

Orientadora:
ALINE MARINS PAES CARVALHO

Coorientador:
ALEXANDRE PLASTINO

NITERÓI

2023

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

M929l Moura, Ricardo de Souza
Less is more: pruning BERTweet architecture for finetuning in twitter sentiment analysis / Ricardo de Souza Moura. - 2023.
73 f.: il.

Orientador: Aline Marins Paes Carvalho.
Coorientador: Alexandre Plastino.
Dissertação (mestrado)-Universidade Federal Fluminense, Instituto de Computação, Niterói, 2023.

1. Mineração de opiniões (Computação). 2. Processamento de linguagem natural (Computação). 3. Twitter (Site de relacionamentos). 4. Produção intelectual. I. Carvalho, Aline Marins Paes, orientadora. II. Plastino, Alexandre, coorientador. III. Universidade Federal Fluminense. Instituto de Computação.IV. Título.

CDD - XXX

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

RICARDO DE SOUZA MOURA

LESS IS MORE: PRUNING BERTWEET ARCHITECTURE FOR FINETUNING IN TWITTER SENTIMENT ANALYSIS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação

Aprovada em Março de 2023.

BANCA EXAMINADORA

Profa. ALINE MARINS PAES CARVALHO - Orientadora, UFF

Prof. ALEXANDRE PLASTINO - Coorientador, UFF

Profa. FLAVIA CRISTINA BERNARDINI, UFF

Prof. DIEGO FURTADO DA SILVA, ICMC/USP

Niterói

2023

I dedicate this work to my children, who encourage me to fulfil my aspirations and inspire me to become a better human every day.

Agradecimentos

To my advisors, who showed me the paths to follow and for the trust placed in me. To my ex-wife (Michelle), children (Matheus and Clara) and current girlfriend (Márcia) for having understood the importance of this achievement and encouraging me to carry it out. And finally, to my mother (Romany) and father (José Emílio - in memory), for all their effort and dedication in providing me with a solid foundation of education and character.

Resumo

Com o auxílio de técnicas de aprendizado por transferência e mecanismos de atenção, modelos baseados em *Transformer*, como o BERT, alcançaram resultados no estado-da-arte para várias tarefas de Processamento de Linguagem Natural (PLN). O tamanho desses modelos costuma ser dimensionado em termos do número de parâmetros para contabilizar a absorção de mais informações. No entanto, vários estudos têm chamado a atenção para sua sobreparametrização e os custos de experimentar modelos tão grandes. Esta dissertação investiga a sobreparametrização do BERTweet, um modelo baseado em *Transformer* treinado com dados do Twitter, com foco na tarefa de análise de sentimento de tweets. A dissertação contribui com um método de poda que visa reduzir o tamanho do BERTweet antes de ajustá-lo à tarefa final. Os experimentos avaliam vários modelos podados propostos que, após o procedimento de ajuste fino, alcançam desempenho ainda superior ao do modelo completo com o mesmo procedimento de ajuste fino realizado. Depois de aplicar o método no BERTweet, o modelo significativamente podado com melhor resultado geral de desempenho preditivo foi o resultado da poda de 47,22% de todas as *heads* (68 de 144 *heads*), que são os componentes que computam em paralelo os valores de atenção. Na verificação de generalização, o tempo gasto para o ajuste fino deste modelo foi reduzido em pelo menos 10%, alcançando o mesmo ou melhor desempenho preditivo que o do modelo completo considerando um nível de significância de 0,05. O método proposto também pode ser aplicado a outros modelos ou tarefas baseados em Transformer para encontrar modelos compactados com desempenho semelhante ao completo. Inesperadamente, durante a execução de nosso método, nos deparamos com outro modelo ainda mais podado (74,31% – 107 de 144 *heads*) com um elevado desempenho preditivo que, embora tenha sido produzido pelo uso parcial de nosso método, pode ser adotado e investigado em trabalhos futuros.

Palavras-chave: Análise de Sentimentos , twitter , modelo de linguagem , Transformer , compressão de modelo, finetuning

Abstract

With the aid of transfer learning techniques, transformer-based models such as BERT have reached state-of-the-art results for several Natural Language Processing (NLP) tasks. The size of those models has been scaled in terms of the number of parameters to account for absorbing more information. However, several studies have called attention to their overparametrization and the costs of experimenting with such huge models. This dissertation investigates the overparametrization of BERTweet, a transformer-based model trained with Twitter data, focusing on the prevalent task of tweets sentiment analysis. The dissertation contributes with a pruning method that aims at reducing BERTweet size before tuning it to the task. The experiments evaluate several proposed pruned models that, after the finetuning procedure, achieve even superior performance than when tuning the complete model. After applying the method on BERTweet, the significantly pruned model with the best overall predictive performance was the result of pruning 47.22% of all heads (68 from 144 heads). In the generalization check, the time spent to finetuning this pruned model was reduced by at least 10% while achieving the same or better predictive performance than the original model with a significance level of 0.05. The proposed method can also be applied to other transformer-based models or tasks to find compressed models that performs similarly to the complete one. Unexpectedly, during the execution of our method, we came across another model that is still more pruned (74.31% – 107 from 144 heads) with high predictive performance that, although the partial use of our method has produced it, can be adopted and researched in future works.

Keywords: sentiment analysis , twitter , language model , transformer , model compression , finetuning

List of Figures

List of Tables

Contents

1 Introduction

Over the past years, the number of internet users exploring social media platforms such as Twitter[1] has progressively increased. Twitter is one of the most popular social microblogging platforms, with an average of 6,000 posts published every second, corresponding to over 500 million posts per day[2]. Users can share real-time information about all topics and events on Twitter. However, unlike other social media networks, Twitter content – the tweets – is characterized by a unique and informal linguistic style, where users frequently use misspelled words, slang, hashtags, emoticons, and share URL links in a limited space of 280 characters. The tweets' language and lack of context in such small texts make it challenging to discover helpful information even with modern Machine Learning and Natural Language Processing (NLP) methods.

One of the most popular tasks devised to categorize tweets is sentiment analysis. Sentiment analysis, or opinion mining, is the field of study that aims to extract opinions, sentiments, emotions, moods, and attitudes from natural language texts using computational methods (LIU, 2020). Generally, the sentiment analysis task is simplified to figure out texts polarity or valence, i.e., to detect if they carry a positive or negative connotation.

A trending research topic in NLP, including Twitter sentiment analysis, is how to numerically represent textual content for a machine to deal with it. The traditional Bag-of-Words (BoW) (TURNEY; PANTEL, 2010) method represents words based on their frequency in a corpus. However, it suffers from sparsity and hence the curse of dimensionality due to the large number of unique words a corpus may contain. Moreover, it fails to represent the semantics of words. To address those issues, neural-based learning techniques induce *embeddings* – dense real-valued low dimensional vectors – from large corpora to represent words and texts.

Pioneering strategies for generating embeddings (MIKOLOV et al., 2013; TURNEY; PANTEL, 2010; PENNINGTON; SOCHER; MANNING, 2014) adopted static vector representations

[1] http://www.twitter.com
[2] https://www.internetlivestats.com/twitter-statistics/

for words, which remain the same regardless of the context they appear. More recently, contextualized embedding techniques, such as BERT (DEVLIN et al., 2019), ELMo (LIU, Yijia et al., 2020), and ULMFiT (HOWARD; RUDER, 2018), emerged aiming mainly at the existence of many possible meanings for a word. This way, if one employs the same word with different meanings, it can be represented with different vectors depending on the context it appears.

Contextualized embeddings are usually induced with large neural networks trained with an extensive set of self-supervised examples. BERT, for example, is trained on a large corpus of unlabelled conventional texts, including English Wikipedia and the Book Corpus (800M words). BERT architecture is a stack of Transformer encoder layers (VASWANI et al., 2017), each composed of two sublayers linked to each other: a self-attention layer with parallel heads and a Feed Forward Neural Network (FFNN) layer. The neural network designed to learn contextualized embeddings also targets transfer learning (PAN; YANG, 2010; RUDER et al., 2019), as they allow for employing the embeddings learned from generic tasks and data to specific tasks and their examples. In this case, a pre-trained model is either adapted with a stage of continuous pre-training or *finetuned* for a specific task and target dataset (GURURANGAN et al., 2020). Such strategies have been broadly adopted and rapidly advanced state-of-the-art results for distinct NLP tasks, such as text classification, machine translation, named entity recognition, among many others (RUDER et al., 2019; HAN; PANG; WU, 2021).

However, although BERT-based models have achieved state-of-the-art results in several NLP tasks, representing tweets remains a challenge. In this context, noting the specificities of tweets and their importance for discovering helpful information from user-generated context in many domains, Nguyen, Vu, and Nguyen (2020) developed BERTweet, a BERT model pre-trained exclusively from tweets in English.

Despite their remarkable performance, BERT-like models, including BERTweet, still lack theoretical results that could better explain the reasons for their performance, limiting further improvements of their architectures. Furthermore, several other Transformer-based models have emerged since BERT's launch, each with more parameters than the other, expecting to retain much more knowledge. Notwithstanding the successful performance achieved by these huge models, previous studies have demonstrated they are usually overparameterized, retaining redundant knowledge, and requiring expensive hardware with possible environmental consequences to train them (KOVALEVA et al., 2019; VOITA

et al., 2019; BENDER et al., 2021). Such observations have encouraged research on compressing their architectures without considerably harming their performance.

This way, recent studies (PRESS; SMITH; LEVY, 2020; K et al., 2020; MICHEL; LEVY; NEUBIG, 2019; PRASANNA; ROGERS; RUMSHISKY, 2020) have conducted computational experiments to evaluate the predictive performance of compressed models using specific datasets and for limited tasks. For instance, Michel, Levy, and Neubig (2019) have noticed that pruning parts of the models did not incur any noticeable negative impact. Moreover, Prasanna, Rogers, and Rumshisky (2020) have observed that more than one sub-network performs as well as the complete network. However, they did not provide any procedure to identify a sub-network better than the complete one. Nonetheless, there is still a gap in understanding how compressed language models perform on the sentiment analysis task on tweets. Moreover, to the best of our knowledge, there is no widespread method in the literature that aims to suggest a highly compressed Transformer-based model that could be applied to the sentiment classification of tweets achieving similar or even better performance than the complete model.

1.1 Research Questions

We hypothesise that there are compressed models that can achieve as good predictive performance as the complete model, or even better, in the sentiment analysis task. With this in mind, we aim to identify at least one significantly compressed of these. Thus, in this dissertation, we intend to answer the following research questions (RQ):

RQ1 : Are there finetuned BERTweet pruned models that achieve competitive or even better predictive performance than the finetuned complete model in the tweets sentiment analysis task?

RQ2 : If so, how could one discover a significantly pruned model that reaches such performance?

To answer these questions, this dissertation presents a method to efficiently compress Transformer-based models, focusing on finetuning them to be used in the sentiment classification of tweets. Since the self-attention mechanism is one of the fundamental underlying components of Transformer-based models and based on the results achieved by Kovaleva et al. (2019), which demonstrates that disabling some heads could even increase the performance, our proposed method focuses on pruning BERTweet heads, resulting in

a more compact model to be finetuned. The main goal is to achieve a predictive performance close to or even better than the finetuned complete model. Furthermore, a pruned model potentially requires less memory usage than its complete version, less space in the disk to be stored, and less computational time to be finetuned. All these advantages are levered to the finetuning procedure and at the inference time. To evaluate the proposed compressing method, we perform computational experiments by exploring BERTweet's architecture using a significant set of twenty-two datasets of tweets (CARVALHO; PLASTINO, 2021).

Using the proposed compression method, we could efficiently remove 68 heads from the original model, corresponding to 47.22% of all heads. This pruned model achieved similar or even better predictive performance than the complete model in all evaluation datasets. Moreover, the size of the model was reduced by 13.27%, and the time spent in finetuning decreased by at least 10%, representing at least 30 watts consumption reduction per hour using the hardware we have employed (Tesla P100-SXM2 GPU).

Surprisingly, while evaluating our method against a random order strategy, we came across a compressed model created from pruning 107 heads from the original model that achieved similar or even better predictive performance than the complete model in all evaluation datasets. This prune corresponds to 74.31% of all heads, which reduces 19.10% from the original model size. Furthermore, the time spent in finetuning decreased by at least 52%, representing approximately 150 watts consumption reduction per hour using the hardware we have employed (Tesla V100-SXM2 GPU).

Given the accelerated rate that new tweets arrive daily, a model may be finetuned several times when in operation. This way, we believe the amount of resource reduction is notable for those intending to finetune BERTweet with their own Twitter sentiment classification dataset and have limited computational resources but aim at acquiring similar or superior predictive performance to the original model.

1.2 Contributions

To sum up, this dissertation contributes with:

1. A compressing method applied on BERTweet to the sentiment classification task, which can be extended to other Transformer-based models and tasks.

2. A methodology to evaluate the finetuning procedure on compressed models.

3. An analysis of a large set of twenty-two datasets of tweets that have been extensively used in the literature of Twitter sentiment analysis.

4. A map of heads to be pruned from the BERTweet model according to the importance order estimated by the method.

5. A map of heads to be pruned from the BERTweet model according to an importance order chosen by chance.

1.3 Organization

This dissertation is organized as follows. Chapter 2 presents an overview of BERT's architecture, the self-attention mechanism, and the BERTweet model. In Chapter 3, we present some related works. The proposed compressing method and the computational experiments [3] performed to evaluate it are detailed in Chapters 4 and 5, respectively. Finally, in Chapter 6, we present the conclusions and directions for future research.

[3]The code and experiments are publicly available https://github.com/MeLLL-UFF/multiheads_sentiment

2 Key Concepts

In this chapter, we introduce the concepts necessary for understanding the compressing method proposed in this dissertation and presented in Chapter 4. Section 2.1 gives an overview of BERT's architecture model. We further detail the self-attention mechanism in Section 2.2 as it is an essential component of Transformer-based models and our proposed compressing method. Lastly, Section 2.3 presents specific aspects of BERTweet.

2.1 BERT Overview

In their paper, Vaswani et al. (2017) introduced the Transformer, a sequence transduction model based entirely on attention mechanisms (BAHDANAU; CHO; BENGIO, 2015; LUONG; PHAM; MANNING, 2015). Its architecture comprises a stack of encoder layers and a stack of decoder layers. Each stack has its corresponding embedding layer for its respective inputs, with an output layer on top of the decoder stack. The encoder embedding layer feds data to the first encoder layer, but from the second layer, the input data is the output data from the previous layer. Similarly, the decoder embedding layer feds data to the first decoder layer, but from the second layer, the input data is the output data from the previous layer; however, all decoder layers receive the output from the last encoder layer as input as well. Each encoder layer comprises two sublayers linked to each other: a self-attention layer with multiple parallel heads and a Feed Forward Neural Network (FFNN) layer. Each decoder layer, in turn, comprises three sublayers linked to each other: a self-attention layer with multiple parallel heads, an encoder-decoder attention layer with multiple parallel heads, and an FFNN layer. The purpose of the encoder stack is to produce an encoded representation of the input sequence. On the other hand, the decoder stack seeks to deliver a target sequence aimed at the transduction of the input sequence.

BERT is a stack of Transformer-encoder layers designed by Devlin et al. (2019) and pre-trained on unlabeled data over two different tasks: Masked Language Model-

ing (MLM) and Next Sentence Prediction (NSP). The data used to pre-train the model include the BooksCorpus (ZHU et al., 2015) (800M words) and the English Wikipedia (2,500M words) corpora. In MLM, some tokens from a sequence are masked, and the model has to predict an appropriate token to fill that mask with. In NSP, in turn, two masked sentences are concatenated as input. Sometimes they correspond to sentences next to each other in the original text, sometimes not. The model then has to predict whether the two sentences follow each other.

Devlin et al. (2019) released BERT models in two different sizes: $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$. While $\text{BERT}_{\text{BASE}}$ consists of 12 encoder layers, each with 12 heads in its self-attention sublayer, $\text{BERT}_{\text{LARGE}}$ consists of 24 encoder layers, each with 24 heads in its self-attention sublayer.

BERT receives as input a single sentence (for single-sequence tasks like sentiment analysis) or a pair of sentences (for sequence-pair tasks like question-answering) represented as a sequence of tokens. The WordPiece algorithm (WU et al., 2016), pre-trained with a vocabulary of 30,000 tokens, tokenizes the input.

Devlin et al. (2019) also proposed a framework to finetune BERT pre-trained model using labeled data for a specific downstream task to leverage and refine the knowledge acquired during the pre-training process. In addition, they presented a feature-based approach, where fixed features are extracted from the pre-trained model to define embedding representations for tokens. Next, those embeddings can serve as examples to train any classification model.

2.2 The Self-Attention Mechanism

Words with the same spelling can have different meanings depending on the sentence they appear in or their position within the sentence. The main goal of the Transformer self-attention mechanism is to compute a contextualized word embedding that leverages the importance of other words[1] in the same sentence.

The self-attention mechanism follows the attention mechanism (BAHDANAU; CHO; BENGIO, 2015; LUONG; PHAM; MANNING, 2015), epitomized by Vaswani et al. (2017) and formulated as follows:

[1] Although we refer to each sentence piece as "words", the process is performed for all tokens generated by the tokenization procedure.

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.1}$$

where Q, K, and V are matrices composed of query, key, and value vectors, respectively. The query and key vectors have the same dimension d_k, whereas the value vector has dimension d_v. The attention mechanism aims at calculating weights that represent the alignment of the inputs (*key*) relative importance in the sequence for a particular input (*query*). More generally, Q are the vectors one wants to calculate attention for, while K are the vectors one wants to figure the attention against. V, in turn, is used to represent these attentions after being weighted and summed up. The difference between attention and self-attention is that the latter assumes the key and query are in the same sentence.

The attention operation performs in parallel for each head from a layer. This way, the output of that layer must be a composition of the output from each of its heads. To this end, Vaswani et al. (2017) proposed to concatenate those outputs, as shown in Equations 2.2 and 2.3:

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^0 \tag{2.2}$$

$$\text{head}_i = \text{Attention}(QW_i^Q,KW_i^K,VW_i^V), \tag{2.3}$$

where h is the number of heads and W^0, W_i^Q, W_i^K, and W_i^V are weight matrices optimized during the training process.

BERT contains a stack of encoder layers with self-attention sublayers. The input from the first layer is a matrix compound of vectors looked up from the static vocabulary of each token from the input sequence. In the other layers, the keys, values, and queries of the self-attention sublayer come from the output of the previous layer (VASWANI et al., 2017). Therefore, Equations 2.2 and 2.3 can be simplified to Equations 2.4 and 2.5, as follows:

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^0 \tag{2.4}$$

$$\text{head}_i = \text{Attention}(XW_i^Q,XW_i^K,XW_i^V), \tag{2.5}$$

where X is a matrix compound of the output from the previous layer.

To aid the understanding, we present a simple example of an input sequence composed of three tokens: A, B, and C. Next, we describe the process that occurs in one head for the first input token, A, but the same applies to all input tokens from the input sequence.

As the first step, for each token, A, B, and C, which are represented by input vectors x_A, x_B, and x_C (output from the previous layer), three other vectors are defined: query vectors: q_A, q_B, and q_C; key vectors: k_A, k_B, and k_C; and value vectors: v_A, v_B, and v_C. They are defined by multiplying the input token representations x_A, x_B, and x_C to the matrices (W^Q, W^K, and W^V) learned during the training process.

The second step consists in computing a score s_{ij} for each input token i regarding input token j. This score indicates how much attention should be placed on other parts of the input sentence. It is calculated as the dot product between vector q_i, from input i, and vector k_j, from the input j, i.e., $s_{ij} = q_i \cdot k_j$. Considering the given example and token A, after calculating the scores s_{AA}, s_{AB}, and s_{AC}, if $s_{AB} < s_{AC}$, it means that token C influences the representation of token A more than token B.

The third step consists in (i) computing the ratio between those scores and the square root of the dimensionality of the key vector (d_k), which in BERT is 64, and (ii) applying a softmax operation on these results. Assuming that $s_{AA} = 110$, $s_{AB} = 50$, and $s_{AC} = 100$, then the results of the softmax operation are 77.70%, 0.04%, and 22.26%, respectively. These results express how relevant an input token from the sequence regards the input token A. Note how small the softmax result is for a token B (0.04%).

Next, in the fourth step, the value vectors v_A, v_B, and v_C are multiplied to their respective softmax results, producing the weighted value vectors wv_{AA}, wv_{AB}, and wv_{AC}. The intuition here is to keep the highest value vectors to more relevant input tokens and drop irrelevant ones once they are multiplied to tiny numbers.

In the fifth and final step, the weighted value vectors are summed up, resulting in a vector z_A, i.e., $z_A = wv_{AA} + wv_{AB} + wv_{AC}$. Steps 1 to 5 occur in each head that compounds the self-attention sublayer. The z_A vectors from each head are concatenated and multiplied to W^0 (another matrix optimized during the training process) so that the FFNN sublayer receives only one vector. It is then sent to the FFNN sublayer, which passes its output to the next encoder layer.

2.3 BERTweet Language Model

Tweets contain a unique and informal linguistic style, where users frequently use misspelled words, slang, hashtags, emoticons, and share URL links. This way, existing language models trained on conventional text corpora, like Wikipedia, may not correspond well with tweets. Models trained on such corpora hold a traditional vocabulary that scarcely matches the tweet vocabulary, even if applying a data-wrangling process. Even in Transformer-based architectures that adopt subwords-based tokenizers, the noisy vocabulary of tweets may break words into several small pieces, hence completely losing the original sentence meaning.

To fill that gap, Nguyen, Vu, and Nguyen (2020) trained BERTweet, a public large-scale pre-trained language model for English tweets. Its architecture is based on $\text{BERT}_{\text{BASE}}$ (12 layers and 12 heads in each layer). It was trained based on the RoBERTa (LIU, Yinhan et al., 2019) pre-training procedure, which, when compared to BERT, includes: training the model longer on more data using bigger batches, eliminating the next sentence prediction target task; training on longer sequences; and dynamically changing the masking pattern applied to the training data. The corpus used to train BERTweet consists of 850M English tweets, which contain 16bn tokens and require about 80GB of drive space. From the 850M tweets, 845M were streamed from 01/2012 to 08/2019, and 5M are related to the COVID-19 pandemic. BERTweet models are publicly available in Huggingface (WOLF et al., 2020).

BERTweet addressed three NLP tasks for Twitter data in (NGUYEN; VU; NGUYEN, 2020): Part-Of-Speech (POS) tagging, Named-Entity Recognition (NER), and text classification. The results show that BERTweet outperformed the previous state-of-the-art models on all these tasks.

3 Literature Review

The number of parameters of transformer-based models has grown continually over the years. BERT$_{BASE}$, for example, has 110M parameters whereas GPT-2 (RADFORD et al., 2018), Turing-NLG (ROSSET, 2020), and GPT-3 (BROWN et al., 2020) have 1.5B, 17B, and 175B, respectively. Besides the inaccessible amount of parameters to low computational resources, experiments conducted in several previous works showed that pruning some heads or even layers does not harm the model's performance significantly and may still increase it in some cases(MICHEL; LEVY; NEUBIG, 2019; VOITA et al., 2019; SAJJAD et al., 2020). Thus, interest in efficiently compressing models has increased, focusing on smaller models with minimum performance decline compared to larger models. They can be categorized into three main approaches: Knowledge Distillation (HINTON; VINYALS; DEAN, 2015), which is the process of transferring knowledge from a large model to a smaller one, Quantization (SHEN et al., 2020; ZAFRIR et al., 2019), in which parameters are represented with fewer bits, and Pruning (KOVALEVA et al., 2019; K et al., 2020; MICHEL; LEVY; NEUBIG, 2019; VOITA et al., 2019; SAJJAD et al., 2023; CLARK et al., 2019; KAO et al., 2020; BAAN et al., 2019; TENNEY; DAS; PAVLICK, 2019; GORDON; DUH; ANDREWS, 2020; FAN; GRAVE; JOULIN, 2020; CHEN et al., 2020; SANH; WOLF; RUSH, 2020), which involves disabling or removing parts of the networks that are either redundant or less relevant to the task at hand. Pruning approaches are categorized into unstructured and structured pruning. Unstructured methods prune weights irrespective of which architecture block they belong to, whereas structured methods prune architecture blocks. Our work fits into the structured pruning category once we propose a method to efficiently prune heads and layers from BERTweet. Next, we briefly describe the methods according to the moment the pruning happens (before or after training). We also include methods that prune models focused on analyzing them.

3.1 Pruning after training

Fan, Grave, and Joulin (2020) have proposed the *LayerDrop*, a form of structured dropout, which has a regularization effect during training and allows for efficient pruning at inference time. K et al. (2020) have experimented with a varying number of layers, heads, and parameters from the mBERT model (DEVLIN et al., 2019), the multilingual version of BERT. They concluded that the number of heads was less significant than the number of layers.

Michel, Levy, and Neubig (2019) evaluated pruned models at test time. They removed one or more attention heads from a given architecture and measured the performance difference at test time. They considered two trained models: WMT (VASWANI et al., 2017), reporting BLEU (BiLingual Evaluation Understudy) scores on the *newstest*2013 test set, and BERT (DEVLIN et al., 2019), reporting accuracies on MultiNLI (WILLIAMS; NANGIA; BOWMAN, 2018) validation set. They modified the multi-headed attention formula by adding a new variable to zero out some head computations. The first experiment removed the heads one by one and checked the importance of a head, comparing the performance of the complete model against the performance of the pruned one. If the performance was similar, the head was marked as redundant given the rest of the model. They concluded that, at test time, most of the heads were redundant given the rest of the model. The second experiment removed all heads except one in a given layer and compared the performance of the complete model against the pruned one. They figured out that, for most layers, one head is indeed sufficient at test time. Lastly, the third experiment established an order of importance for all heads, removing them individually to evaluate what could happen when removing heads from different layers. They noticed that this last approach pruned up to 20% and 40% of heads from WMT and BERT models, respectively, without incurring in any noticeable negative impact. They have also observed that a pruned model is more efficient than the complete one, showing a speedup in inference time by $\approx 17.5\%$ for larger batch sizes using a BERT model with 50% of all attention heads pruned.

Sajjad et al. (2023) explored strategies to drop entire layers from pre-trained models. They pruned up to 40% layers from BERT, RoBERTa, and XLNet while keeping up to 98% of the original performance. During this process, they realized that some tasks are more robust to dropping any layers, but, in general, the lower layers are the most crucial to maintaining the downstream task performance. They also stated that the layers must

be dropped regarding the target class to achieve a smaller model close to the optimal network.

3.2 Pruning before or during training

Most studies regarding pruning approaches consider compressing the model *after* training, making inference more efficient. However, Frankle and Carbin (FRANKLE; CARBIN, 2019) investigated if training a pruned model instead of a complete one could reach a suitable model in less time. As a result, they concluded that exist smaller sub-networks trained from scratch faster than their larger counterpart, achieving similar or even higher accuracy. Prasanna, Rogers, and Rumshisky (2020) and Chen et al. (2020) analysed BERT from the perspective of Frankle and Carbin's hypothesis(FRANKLE; CARBIN, 2019). Prasanna, Rogers, and Rumshisky (2020) showed that it is possible to uncover sub-networks that perform as well as the complete model. Moreover, they noticed that, in many cases, even the "bad" sub-networks could be reinitialized to the pre-trained BERT weights and fine-tuned to achieve robust performance. Chen et al. (2020) focused on finding compressed models that were universally trainable on several downstream tasks. Using an unstructured pruning method, they realized that, on most downstream tasks, the uncovered finetuned models are not helpful to other tasks. Nevertheless, sub-networks discovered from finetuning using the masked language model task are universal and valuable to other tasks. Gordon, Duh, and Andrews (2020) analyzed BERT transfer capability and concluded that it could be pruned before becoming publicly available without affecting its capacity for any tasks.

3.3 Pruning for model analysis

Pruning processes have also been used to analyze models. The intuition is that a compressed model comprises the most valuable components for prediction. This way, uncovering what these sub-networks do might reveal what the complete network does. Kovaleva et al. (2019) focused on the interpretation of self-attention, which is one of the fundamental underlying components of BERT. They proposed five attention patterns that repeat across different heads, indicating the overall model overparameterization. They show that disabling attention in certain heads improves performance over the regular finetuned BERT models. Voita et al. (2019) performed computational experiments to analyze the influence of individual attention heads of transformer models on machine translation. Moreover,

they investigated if the quality of the translation could be kept even with a reduced number of attention heads. For this purpose, they first identified the most critical attention head from each encoder layer, using the layer-wise relevance propagation technique (DING et al., 2017) and characterized roles for them, creating three distinct types: positional, syntactic, and rare words. To understand if the remaining heads (which are not part of any role) are still essential or redundant for translation, they introduced a head pruning process based on the technique proposed in (LOUIZOS; WELLING; KINGMA, 2018). The computational experiments showed that most attention heads could be removed from the fully trained model without significantly losing translation quality. However, they reported not being confident in ensuring that a model could be trained from scratch with such few heads.

Final Remarks In the current dissertation, we propose a pruning method and analyze the performance of pruned BERTweet models for the sentiment analysis task on Twitter data. Presumably, having a huge model to be applied to tweets seems excessive, once tweets are too concise. Our method prunes the model head by head according to an order of importance calculated using an approach based on the method proposed in (MICHEL; LEVY; NEUBIG, 2019). However, different from (MICHEL; LEVY; NEUBIG, 2019) and others (FAN; GRAVE; JOULIN, 2020; K et al., 2020; KOVALEVA et al., 2019; VOITA et al., 2019; SAJJAD et al., 2023), which focus on making inference more efficient, we are also interested in making finetuning more efficient. This way, we would like to know whether there exist pruned models that could be finetuned, consuming less computing resources while still achieving similar or even better performance than finetuning the complete model. Moreover, we provide a model architecture template that one can easily adopt to finetune using their own dataset and reach better results than with the complete model. Our conclusion aligns with the outcomes of (PRASANNA; ROGERS; RUMSHISKY, 2020; CHEN et al., 2020). Although they have used distinct pruning processes, they stated that there are sub-networks that perform as well as the complete model.

4 A method to prune BERTweet heads

Usually, BERT-based pre-trained models are finetuned to be applied to a downstream task. This dissertation presents a method to prune heads of BERTweet pre-trained models, focusing on the finetuning performance applied to the task of Twitter sentiment analysis. The main goal is to reach a smaller model that achieves a competitive or even better predictive performance than the complete model. We argue that a pruned model would require less memory usage than its complete version, less space in the disk to be stored, and its finetuning process would take less computational time. Hence, the pruned model would require simpler hardware to be adjusted and embedded in the production system while also contributing to less carbon emission (STRUBELL; GANESH; MCCALLUM, 2019). Notably, in this dissertation, we leverage the possible redundant information that BERTweet heads might accommodate by choosing which heads to remove when finetuning them for tweets sentiment analysis.

However, finding a pruned model that performs as well as the original one is not trivial. Previous studies pointed out that several sub-networks may have the same or better predictive performance as the complete model (FRANKLE; CARBIN, 2019). Specifically, BERTweet, the base architecture that we select to prune, comprises 144 heads distributed in 12 layers. This way, to avoid the combinatorial explosion of testing each one of the vast numbers of models that could be generated when pruning subsets of BERTweet heads, the proposed method establishes an order of importance for the heads and prunes them incrementally in the opposite order. In this context, the method works in two sequential steps. First, it defines a hierarchy of importance for all heads (called here as STEP 1). Then, considering this hierarchy, it determines the best-pruned model architecture to be finetuned among all pruned models generated during the pruning process (called here as STEP 2). Sections 4.1 and 4.2 detail these steps.

For robustness and to avoid bias, it is crucial to use a different and unique collection of datasets in each of these steps. Moreover, previous literature observed that the finetuning procedure could produce substantially different results due to some random

internal procedure (DODGE et al., 2020). This way, to mitigate the impact it could make on the results achieved by a unique execution, each finetuning procedure executed in the proposed method (Step 1 and Step 2) must be performed a reasonable amount of times using different seeds. Consequently, the model evaluation must be computed according to the mean results achieved by all the executions. We incorporate those experimental decisions into the proposed method to ensure the intended robustness.

Although this study focuses on the BERTweet model and the sentiment classification task, the proposed method is general enough to be applied to other Transformer-based models and tasks.

4.1 Step 1: defining an importance order for the heads

To establish the order of importance for the heads, we propose an approach similar to that presented by Michel, Levy, and Neubig (2019). There, they modify the original Attention Equation 2.5 by including head mask variables (ξ), as one can see in Equation 4.1. The variables ξ constitute a matrix of all heads from all layers. While in the original work ξ could assume values in {0,1} to indicate if a head would be discarded or not, in our approach, they are always 1. This way, all heads are constantly considered in the model optimization, and the related head gradients values are accumulated to compute the numerical importance of the heads.

$$\text{head}_i = \xi_i \text{Attention}(XW_i^Q, XW_i^K, XW_i^V), \tag{4.1}$$

Algorithm 1 details how our proposal determines the importance order of the heads. It receives five input parameters: *datasets*, *seeds*, *nepochs*, *nlayers* and *pretrained_model*. The *datasets* parameter is a set of labeled datasets that are used to define the importance order based on the update of attention weights during the finetuning procedure. The *seeds* parameter is a set of integer numbers used as random seeds in random procedures. They are used in the finetuning process to account for the random weight initialization of the final classification layer and provide reproducibility. The *nepochs* parameter indicates the number of times the finetuning procedure is executed with the same data. The *nlayers* parameter represents the number of layers that compose the architecture. Lastly, the *pretrained_model* is the initial model (pre-trained BERTWeet model in our case), which will be finetuned during the process. For each dataset and each seed (lines 4 and 6),

the procedure first initializes the random seed (line 8) and loads the pre-trained model (BERTweet pre-trained model), adding a classification layer on top of it (line 9). Next, it calls the function responsible for calculating the importance of heads considering a combination of dataset and seed (line 10), which is presented in Algorithm 2. Ultimately, it defines the order of importance of all heads based on the accumulated gradient values from ξ for all the examples, averaged by the datasets, seeds, and epochs (lines 13 and 14).

Algorithm 1 Step 1 - Calculating the importance order of heads

Require:
datasets, *seeds*, *nepochs*, *nlayers*, *pretrained_model*
Ensure:
head_importance_rank
1: *ndatasets* ← |*datasets*|
2: *nseeds* ← |*seeds*|
3: Let *head_importances_by_ds* be a new *ndatasets* × *nseeds* matrix
4: **for** $d = 1$ to *ndatasets* **do**
5: *dataset* ← *datasets*[*d*]
6: **for** $s = 1$ to *nseeds* **do**
7: *seed* ← *seeds*[*s*]
8: Initialize random seed (*seed*)
9: Load *pretrained_model* adding a *classifier* on top
10: *head_importances_by_ds*[*d*,*s*] ← CALCULATE_HEAD_IMPORTANCE (*pretrained_model*, *dataset*, *nepochs*, *nlayers*) ▷ Algorithm 2
11: **end for**
12: **end for**
13: *head_mean_importances* ← *calculate_mean*(*head_importances_by_ds*) ▷ Mean of all datasets, seeds, and epochs results
14: *head_importance_rank* ← order *head_mean_importances* from highest to lowest

Algorithm 2 receives four input parameters: *model*, *dataset*, *nepochs* and *nlayers*. The *model* parameter is the model to finetune. The *dataset* parameter is the labeled dataset used to finetune the model. The *nepochs* and *nlayers* parameters are exactly the same provided to Algorithm 1. For each epoch, the procedure initializes each value in ξ to 1 (line 5). Then, it performs the usual forward step (line 7) and finetunes *model* within the backward pass (line 8) with each batch of examples in the dataset. Next, it accumulates the absolute gradient values computed during the backward pass according to ξ (line 9). The intuition is that the more a variable gradient is absolutely changed, the more important its underlying head is to the finetuned model. At this point, different from Michel, Levy, and Neubig (2019), which normalize the final accumulated gradients results by layer using ℓ_2 normalization, we normalize the accumulated gradients for each batch (line 11) and repeat the normalization at the end of each epoch (line 15) using *Min-Max* normalization.

Algorithm 2 Step 1 - Function for calculating the importance order of heads for one dataset

```
1: function CALCULATE_HEAD_IMPORTANCE(model, dataset, nepochs, nlayers)
2:     Let head_importance be a new nlayers × nlayers matrix    ▷ to accumulate the gradients
3:     ntokens ← number of tokens in dataset
4:     for epoch = 1 to nepochs do
5:         Let head_mask be a nlayers × nlayers matrix filled with 1
6:         for each batch ∈ dataset do
7:             loss ← Forward pass batch on model using head_mask
8:             gradients ← Backward pass(loss)
9:             head_mask_abs_grads ← get_head_mask_abs_grads(gradients)    ▷ gradients from head_mask only
10:            head_importance ← head_importance + head_mask_abs_grads
11:            head_importance ← min_max_normalization(head_importance)
12:            Update model parameters using gradients
13:        end for
14:        head_importance ← head_importance/ntokens    ▷ To avoid the influence of the number of tokens each dataset has
15:        head_importance ← min_max_normalization(head_importance)
16:    end for
17:    return head_importance
18: end function
```

4.2 Step 2: Selecting the best-ranked pruned model

Once the order of importance for the heads is determined, Step 2 uses it as input to search for a significantly pruned model. To execute this step, we propose three different approaches. In the first approach (H1), explained in subsection 4.2.1, we incrementally prune the model's heads following the order of importance determined in Step 1 without any restriction. In this case, the procedure produces several models for each dataset that will be evaluated later. In the second approach (H2), described in subsection 4.2.2, although we also incrementally prune the model's heads, we evaluate the pruned model produced at each iteration and keep the pruning only if the predictive performance of the current pruned model is better than the previous one. Otherwise, we hold the head on the model. The third approach (H3), detailed in subsection 4.2.3, is very similar to the second one; however, instead of evaluating the predictive performance of the pruned model at each iteration against the result of the previous one, the comparison is always made against the predictive performance of the complete model. At the end, the second and third approaches produce just one model for each dataset. Independently of the adopted

approach, we established the strategy of removing the entire layer if we have pruned all the heads in the same layer.

4.2.1 Step 2 – Unrestricted incremental pruning (H1)

Algorithm 3 details the pruning procedure without restricting the heads that can be pruned. It receives five input parameters: *datasets*, *seeds*, *head_importance_rank*, *nheads* and *pretrained_model*. The *datasets* parameter is a set of labeled datasets that are used to finetune each pruned model generated during the process and to train and test classifiers to evaluate those models. The *seeds* parameter is a set of integer numbers used as random seeds in random procedures and to provide reproducibility. They are used to split each dataset into two parts to feed the finetuning and evaluation process. They are also used internally in these processes themselves. The *head_importance_rank* parameter is the order of importance of the heads determined in Step 1. The *nheads* parameter is the total number of heads from the adopted model architecture. Lastly, the *pretrained_model* is the initial model (pre-trained BERTWeet model, in our case), which will be pruned and finetuned during the process.

In this approach, for each dataset (line 4), we iterate over all possible numbers of heads (k) that can be pruned from the pre-trained model (line 6). At each iteration, we look up the k less important heads (line 7) and then call the function PRUN_ADJ_EVAL (line 8), described by Algorithm 4, to prune and adjust the pre-trained model and return the result of its predictive performance. At the first iteration ($k = 0$), we capture the predictive performance of the complete model once any head can be pruned from the pre-trained model. At the second iteration, the head marked as less relevant in Step 1 is pruned, yielding the first pruned model. Next, that head and the second less relevant head are pruned together, yielding the second pruned model and so on. Since BERTweet architecture comprises 144 heads (12 heads for each of the 12 layers), and the proposed method removes heads incrementally, one at each time, we end up with 144 models' predictive performance (one from the complete model and 143 from the pruned ones) for each dataset. Ultimately, we perform a ranking process to determine the best model architecture based on the predictive performances achieved by the evaluated models (line 11). To this end, we call the RANKING_MODELS function, expressed by Algorithm 5.

The PRUN_ADJ_EVAL function, detailed in Algorithm 4, concerns pruning, adjusting, and evaluating a pre-trained model. It receives four input parameters: *pretrained_model*, *dataset*, *seeds*, and *heads_to_prune*. The *pretrained_model* and *seeds* parameters are

Algorithm 3 Step 2 (H1) – Determining the best-pruned model to be finetuned, among all pruned models evaluated during the pruning process

Require:
datasets, *seeds*, *head_importance_rank*, *nheads*, *pretrained_model*
Ensure:
best_pruned_model
1: $ndatasets \leftarrow |datasets|$
2: $nseeds \leftarrow |seeds|$
3: Let $predict\_performances$ be a new $ndatasets \times nheads$ matrix
4: **for** $d = 1$ to $ndatasets$ **do**
5: $dataset \leftarrow datasets[d]$
6: **for** $k = 0$ to $(nheads - 1)$ **do**
7: $heads\_to\_prune \leftarrow get\_first\_kheads(head\_importance\_rank, k)$
8: $predict\_performances[d, k] \leftarrow$ PRUN_ADJ_EVAL(*pretrained_model*, *dataset*, *seeds*, *heads_to_prune*) ▷ Algorithm 4
9: **end for**
10: **end for**
11: *best_pruned_model* ← RANKING_MODELS (*predict_performances*) ▷ Algorithm 5

the same input parameters from Algorithm 3. The *dataset* parameter is a dataset used to adjust and evaluate the pruned model. Finally, the *head_to_prune* parameter is a set of heads to prune from the pre-trained model. In Algorithm 4, for each provided seed (line 4), the dataset is split into two parts (*trains_ds* and *eval_ds*) to adjust and evaluate the model (line 7). Next, the pre-trained model is pruned (line 8) and adjusted (line 9) using the *train_ds* portion. The model is adjusted in a continuous pre-training fashion (GURURANGAN et al., 2020) to accommodate possible changes in the parameters caused by the removed heads. Thus, unlike Step 1, no additional layer is attached to BERTweet architecture. Instead, we further pre-train the models with the Masked Language Model (MLM) intermediate task (DEVLIN et al., 2019). Precisely, we follow the strategy "Within-Task Further Pre-Training" (SUN et al., 2019), where the models are adjusted by applying the same training strategy adopted in the original MLM training but using data from the target task. We keep the same strategy since finetuning and evaluating the models adequately, concerning the target task, take too much time in the available infrastructure. After this procedure, the model is evaluated (line 12) using the *eval_ds* part of the dataset. At this point, we aim to identify the predictive performance of the adjusted pruned model. To this end, every example is passed through the model to extract the features and to create the example representation, namely the *embeddings* (line 10). These embeddings are then used to train and test a classifier, considering that embeddings used to test must not be used to train (line 12). The model's predictive

performance results from testing its respective classifier (line 13), and the return of this function is the mean of the predictive performances achieved for all seeds (line 15).

Algorithm 4 Step 2 - Function for pruning, adjusting and evaluating a model

```
1: function PRUN_ADJ_EVAL(pretrained_model, dataset, seeds, heads_to_prune)
2:     Let F1_metrics be an array of nseeds elements
3:     nseeds ← |seeds|
4:     for s = 1 to nseeds do
5:         seed ← seeds[s]
6:         Initialize random seed (seed)
7:         train_ds, eval_ds ← split_dataset(dataset, seed)
8:         model ← create_pruned_model(pretrained_model, heads_to_prune)
9:         model ← continuous_pre_training(model, train_ds)
10:        features ← extract_features(model, eval_ds)
11:        labels ← get_labels(eval_ds)
12:        metrics ← cross_validation(features, labels, seed)
13:        F1_metrics[s] ← metrics
14:    end for
15:    return calculate_mean(F1_metrics)
16: end function
```

The RANKING_MODELS function, described in Algorithm 5, concerns ranking and deciding which model is the best one among all evaluated. It receives one input parameter, $models\_performance$, a matrix with the predictive performances of all models (pruned and complete ones) for each dataset. This procedure starts ranking these models comparatively according to the predictive performance achieved by each dataset (line 2). The model with the best predictive performance is ranked as 1; the second is ranked as 2, and so on. In the case of a draw, the rank values are summed up and divided among the tied models so that if two models draw in the second place, for example, each of them is ranked as 2.5, i.e., $(2 + 3)/2 = 2.5$. Next, the model's ranks are summed up (line 3) and sorted by the rank sum in decreasing order (line 4). Lastly, the function returns the best model (line 5), represented by that which reaches the lowest rank sum.

Algorithm 5 Step 2 - Function for ranking models

```
1: function RANKING_MODELS(models_performance)
2:     rmodels_ds ← rank_models_by_dataset(models_performance)
3:     rsum_models ← sum_rank_models(rmodels_ds)
4:     models_sorted ← get_models_sort_by_ranksum_desc(rsum_models)
5:     return models_sorted[0]
6: end function
```

Ranking models procedure in details

To illustrate the process, next, we present a hypothetical example in which the input data is composed of the results achieved by running the Algorithm 3 with two datasets A and B. For simplicity, we have considered only three models (nheads=2) by each dataset: the complete model ($k = 0$), a one-head pruned model ($k = 1$), and a two-head pruned model ($k = 2$). Table 1 exposes this hypothetical input data.

The ranking models' procedure starts by calculating the ranks based on each dataset's predictive performance results. The highest predictive performance gets ranked 1; the second highest gets ranked 2, and so on. In case of a draw, a mean of the ranks is applied. Table 2 consolidates these ranks, sorting the results by dataset and descending rank.

Table 1: Hypothetical input data to ranking models procedure

Model	**Dataset**	**Pred. Performance**
Complete	A	97.5
One-head pruned	A	97.5
Two-heads pruned	A	98
Complete	B	97.5
One-head pruned	B	98
Two-heads pruned	B	97

Table 2: Hypothetical input data ranked by predictive performance

Model	**Dataset**	**Rank**
Two-heads pruned	A	1.0
Complete	A	2.5
One-head pruned	A	2.5
One-head pruned	B	1.0
Complete	B	2.0
Two-heads pruned	B	3.0

Lastly, the ranks are grouped by model and summed up. We consider the model with the lowest rank sum the best among all evaluated. Table 3 presents the ranking sum applied to the hypothetical example. As can be observed in this case, the best model is represented by the one-head pruned, which means we should prune just the first head from the complete model based on the order of importance defined in Step 1.

Table 3: Hypothetical input data with the ranks grouped by models and summed up

Model	Rank Sum
One-head pruned	**3.5**
Two-heads pruned	4.0
Complete	4.5

4.2.2 Step 2 – Incremental pruning restricted by the previous predictive performance (H2)

This approach is detailed in Algorithm 6 and receives the same input parameters as the previous one (Algorithm 3). In Algorithm 6, although we also incrementally prune the model's heads, we evaluate the pruned model created at each iteration and keep the pruning only if the predictive performance of the current pruned model is better than the previous one. Otherwise, we preserve the head on the model. Thus, we first identify this set of heads to be pruned for each dataset and store them on *best_heads_to_prune* variable (line 4). To this end, for each dataset (line 6), we iterate over all head's index (line 8) and look up the *k-th* head based on the provided order of importance (line 9). Next, we append this current head to the current best set of heads (line 10) and call the function PRUN_ADJ_EVAL (the same Algorithm 4 used in the approach H1) to get the predictive performance of the pre-trained model pruned by these heads (line 11). As we are interested in the best set of heads, for each iteration, we compare the predictive performance of the current model, *F1_mean*, to the best model so far, *best_F1* (line 12). If the *F1_mean* is greater or equal to *best_F1*, we update *best_heads_to_prune* aggregating the new head (line 14) and update *best_F1* to be *F1_mean* (line 13). After iterating over all datasets, we end up with the best model – more precisely, with the heads that must be pruned from the pre-trained model – for each dataset individually. Hence, to run the RANKING_MODELS (the same Algorithm 5 used in H1) function and decides about the best among all those models (line 24), we need to iterate over the datasets one more time to calculate the predictive performance of each of the identified models over all the datasets.

Algorithm 6 Step 2 (H2) – Determining the best-pruned model to be finetuned, among all pruned models generated during the pruning process

Require:
datasets, *seeds*, *head_importance_rank*, *nheads*, *pretrained_model*
Ensure:
best_pruned_model
1: $ndatasets \leftarrow |datasets|$
2: $nseeds \leftarrow |seeds|$
3: Let $best\_F1$ be an array of $ndatasets$ elements initialized with value 0
4: Let *best_heads_to_prune* be an array of $ndatasets$ elements
5: Let $model\_perfs$ be a new $ndatasets \times ndatasets$ matrix
6: **for** $d = 1$ to $ndatasets$ **do**
7: $dataset \leftarrow datasets[d]$
8: **for** $k = 0$ to $nheads$ **do**
9: $head \leftarrow get\_k\_th\_head(head\_importance\_rank, k)$
10: $temp\_heads = best\_heads\_to\_prune[d] + head$
11: $F1\_mean \leftarrow$ PRUN_ADJ_EVAL ($pretrained\_model$, $dataset$, $seeds$, $temp\_heads$) ▷ Algorithm 4
12: **if** $F1\_mean >= best\_F1[d]$ **then**
13: $best\_F1[d] \leftarrow F1\_mean$
14: $best\_heads\_to\_prune[d] \leftarrow temp\_heads$
15: **end if**
16: **end for**
17: **end for**
18: **for** $i = 1$ to $ndatasets$ **do**
19: **for** $j = 1$ to $ndatasets$ **do**
20: $dataset \leftarrow datasets[j]$
21: $model\_perfs[i,j] \leftarrow$ PRUN_ADJ_EVAL ($pretrained\_model$, $dataset$, $seeds$, $best\_heads\_to\_prune[i]$) ▷ Algorithm 4
22: **end for**
23: **end for**
24: $best\_pruned\_model \leftarrow$ RANKING_MODELS ($model\_perfs$) ▷ Algorithm 5

4.2.3 Step 2 – Incremental pruning restricted by the complete model performance (H3)

This approach is detailed in Algorithm 7. It receives the same input parameters as the first and second approaches (Algorithm 3 and Algorithm 6), and it is very similar to the second one. The main difference concerns the way of deciding whether a head must be part of the best set of heads or not. For each iteration, instead of comparing the predictive performance from the current model to that from the best model so far, as we did in the second approach, the predictive performance of the pruned model is always compared to the predictive performance of the complete model (line 16). For this reason, we first capture the predictive performance from the complete model for each dataset (line 6).

Algorithm 7 Step 2 (H3) – Determining the best-pruned model to be finetuned, among all pruned models generated during the pruning process

Require:
datasets, *seeds*, *head_importance_rank*, *nheads*, *pretrained_model*
Ensure:
best_pruned_model

```
ndatasets ← |datasets|
nseeds ← |seeds|
Let cm_F1 be an array of ndatasets elements
for d = 1 to ndatasets do
    dataset ← datasets[d]
    cm_F1[d] ← PRUN_ADJ_EVAL(pretrained_model, dataset, seeds, ∅)        ▷ Algorithm 4
end for
Let best_heads_to_prune be an array of ndatasets elements
Let model_perfs be a new ndatasets × ndatasets matrix
for d = 1 to ndatasets do
    dataset ← datasets[d]
    for k = 1 to nheads do
        head ← get_k_th_head(head_importance_rank, k)
        temp_heads = best_heads_to_prune[d] + head
        F1_mean ← PRUN_ADJ_EVAL(pretrained_model, dataset, seeds, temp_heads)        ▷ Algorithm 4
        if F1_mean >= cm_F1[d] then
            best_heads_to_prune[d] ← temp_heads
        end if
    end for
end for
for i = 1 to ndatasets do
    for j = 1 to ndatasets do
        dataset ← datasets[j]
        model_perfs[i,j] ← PRUN_ADJ_EVAL(pretrained_model, dataset, seeds, best_heads_to_prune[i])        ▷ Algorithm 4
    end for
end for
best_pruned_model ← RANKING_MODELS(model_perfs)        ▷ Algorithm 5
```

5 Computational Experiments

This chapter presents the experimental results obtained by evaluating the method described in Chapter 4. We describe the experimental methodology we followed in Section 5.1. Then, in Section 5.2, we report and discuss the experimental results with which we answer the research questions RQ1 and RQ2.

5.1 Experimental methodology

This section presents the evaluation of our neural language model pruning proposed method. As mentioned in Chapter 4, the method is arranged into two sequential steps. First, it determines an order of importance for all heads (Step 1), next, it identifies the best-pruned model architecture (Step 2) among all evaluated. Sections 5.2.1 and 5.2.2 detail the method's results, i.e., the outputs of Steps 1 and 2, respectively. In addition, to check if the best-pruned model architecture suggested by the method generalizes well, we propose a generalization check to evaluate it. Section 5.2.3 presents and discusses such an evaluation. Figure 1 visually presents the methodology flow applied to execute the experiments.

Datasets: For robustness and to avoid bias, we use a different and unique collection of datasets in each step of the method and the generalization check. We use a large set of twenty-two datasets of tweets[1] that have been extensively used in the Twitter sentiment analysis literature (CARVALHO; PLASTINO, 2021; BARRETO et al., 2022). They provide a diverse scenario in evaluating the performance of the pruned model architectures in the sentiment classification task. We arrange all these data into three groups (G1 for Step 1, G2 for Step 2, and G3 for Generalization Check), trying to equally distribute the datasets and the total amount of tweets among the three groups.

[1]The datasets are publicly available at https://github.com/joncarv/air-datasets

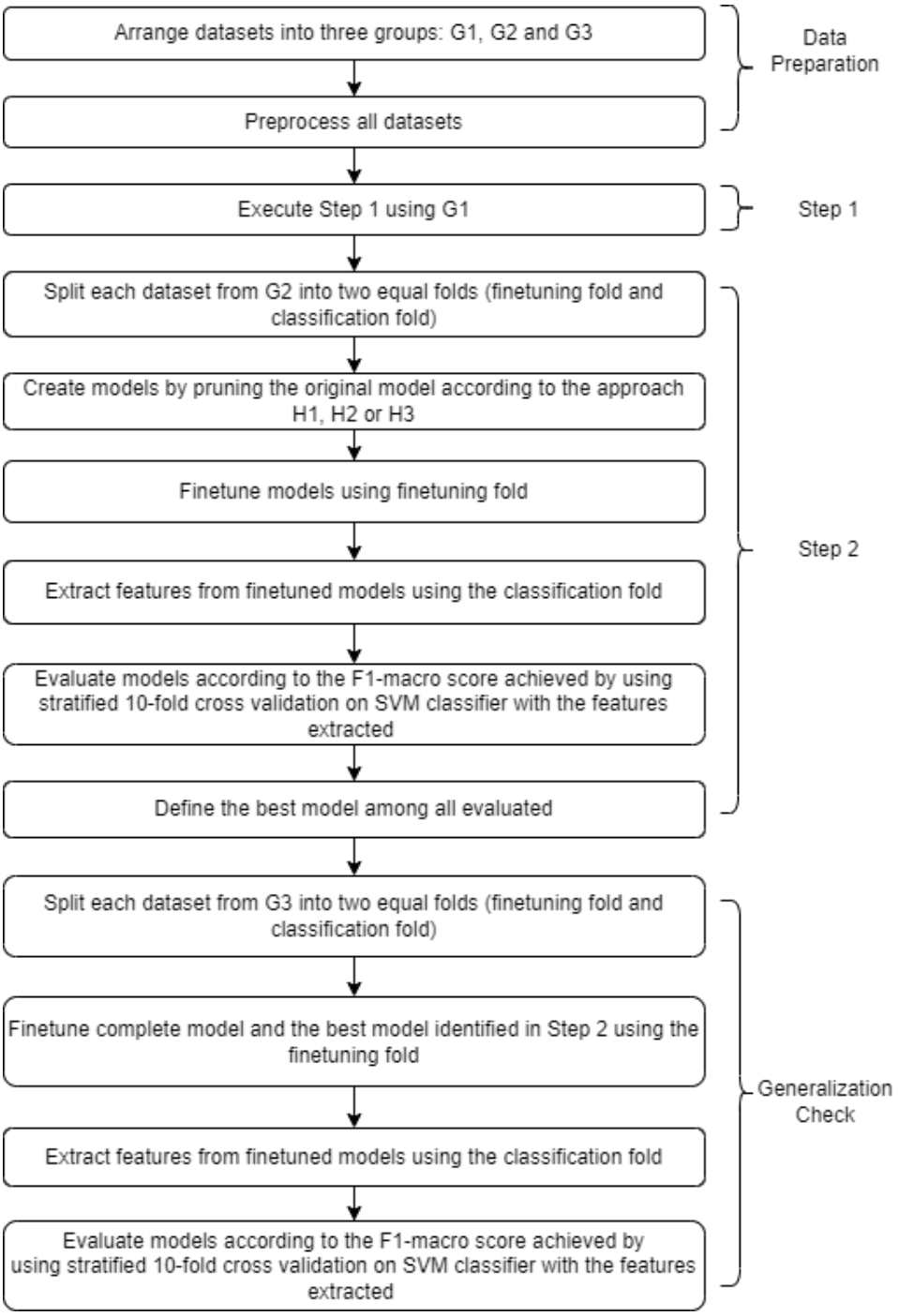

Figure 1: Methodology adopted to evaluate the proposed pruning method

Table 4 presents the datasets used in each group, including their abbreviation name and some statistics, such as the number of positive tweets (*#pos* column), number of negative tweets (*#neg* column), the total number of tweets (*Total* column), the minimum number of tokens, i.e., number of tokens of the smallest tweet in the dataset (*min* column), the maximum number of tokens (*max* column), the mean number of tokens (*mean* column) and its standard deviation (*std* column), and the sum of all tokens from all tweets (*sum* column).

Table 4: Characteristics of the Twitter sentiment datasets ordered by size (Total column)

Dataset	**Abbrev.**	**#pos**	**#neg**	**Total**	**#tokens**				
					min	**max**	**mean**	**std**	**sum**
G1 (Step 1)									
irony	iro	22	43	65	8	46	25.02	8.89	1626
sarcasm	sar	33	38	71	7	46	23.70	8.50	1683
SemEval-Fig	S15	47	274	321	8	49	24.92	7.69	7998
archeage	arc	724	994	1,718	5	47	24.79	8.38	42593
HCR	HCR	539	1,369	1,908	7	76	29.90	6.67	57056
STS-gold	STS	632	1,402	2,034	5	49	21.31	9.08	43348
SemEval16	S16	8,893	3,323	12,216	6	55	28.38	6.65	346740
G2 (Step 2)									
person	per	312	127	439	7	70	27.27	8.36	11973
hobbit	hob	354	168	522	5	53	21.27	9.30	11102
movie	mov	460	101	561	7	57	25.14	9.41	14101
SemEval18	S18	865	994	1,859	3	56	24.01	9.16	44630
SentiStrength	SSt	1,340	949	2,289	5	51	24.44	8.67	55948
Vader	vad	2,897	1,299	4,196	3	52	21.16	8.97	88775
SemEval17	S17	2,375	3,972	6,347	4	53	26.15	7.53	165975
G3 (Generalization Check)									
aisopos	ais	159	119	278	6	43	22.57	8.58	6275
sentiment140	stm	182	177	359	4	52	23.13	9.77	8305
iphone6	iph	371	161	532	7	46	21.43	8.88	11400
sanders	san	570	654	1,224	4	55	24.44	9.13	29917
Narr	Nar	1,739	488	1,227	4	49	21.09	9.32	25872
OMD	OMD	710	1,196	1,906	5	50	22.94	8.18	43715
Target-dependent	Tar	1,734	1,733	3,467	5	76	25.21	8.62	87417
SemEval13	S13	3,183	1,195	4,378	5	92	29.35	7.16	128497

Tweets preprocessing: As this dissertation's main objective is to compare the performance of the finetuning procedure over the original model (complete model) and the pruned ones and not the behavior of different preprocessing techniques, we apply a simple preprocessing strategy as adopted in (BARRETO et al., 2022). More clearly, for each tweet in a given dataset, we only replace URLs by the tag *someurl*, and user mentions by the

tag *someuser*. Lastly, all tweets are lowercased and tokenized, i.e., split into a sequence of tokens. The tokenization procedure follows the same strategy adopted by Nguyen, Vu, and Nguyen (2020) to generate the BERTweet base model "*vinai/bertweet-covid19-base-uncased*". To this end, we leverage the Huggingface Transformer library (WOLF et al., 2020) using the tokenizer with the same name of the model, i.e., "*vinai/bertweet-covid19-base-uncased*".

Computational experiments details:[2] All scripts are coded in Python and are based on PyTorch (PASZKE et al., 2019), Huggingface Transformer library, and scikit-learn (PEDREGOSA et al., 2011). Since all tweets are lowercased, we adopted the BERTweet base model "*vinai/bertweet-covid19-base-uncased*"[3], which results from further pre-training the pre-trained model "*vinai/bertweet-base*" on a corpus of 23M English COVID-19 tweets. While Step 1 and all experiments involving CM and the approach H1 were performed in a Tesla P100-SXM2 GPU within the Ubuntu operating system, running in a machine with Intel(R) Xeon(R) CPU E5-2698 v4 processor, all experiments involving approaches H2 and H3 were performed in Tesla V100-SXM2 GPU within the Centos7 operating system, running in a machine with Intel(R) Xeon(R) Gold 6126.

Computational experiments seeds: Given the importance of executing the finetuning procedure several times, as explained in Chapter 4, we decided to run it ten times. For Step 1 and Step 2, we used seeds *1, 2, 3, 4, 5, 6, 7, 8, 9, 10*. In the generalization check, we used seeds *101, 102, 103, 104, 105, 106, 107, 108, 109, 110*. We establish a different set of seeds for the generalization check to keep it still more isolated from the method execution.

To perform Step 2 of the method, each dataset from G2 is split into two equal parts. The first part, the *finetuning fold*, is used to further adjust each model (the complete BERTweet model or the pruned ones) with the continuous pre-training procedure. The second part, the *classification fold*, is used to learn an SVM classifier (CORTES; VAPNIK, 1995) and evaluate the adjusted models in terms of F1-macro score, following a stratified ten-fold cross-validation. We adopted this classifier once it reached the best overall results in (BARRETO et al., 2022). For each adjusted model and dataset, every tokenized example (a single tweet) is passed through the model to extract the features and create the tweet representation, namely the *tweet embedding*, fed to the SVM classifier. We define the tweet embedding as the mean of each of its token embeddings. The token embedding, in

[2]The code and experiments are publicly available https://github.com/MeLLL-UFF/multiheads_sentiment

[3]https://huggingface.co/vinai/bertweet-covid19-base-uncased

turn, is defined as the concatenation of the weights from the last layers (four or fewer, depending on the number of layers available in the pruned model). Based on the results presented in (BARRETO et al., 2022), we adopted the scikit-learn implementation of the SVM classifier (SVC) (PEDREGOSA et al., 2011) using all default parameter values, except for the class balance parameter ($class\_weight = balanced$).

The generalization check evaluates the selected pruned model and compares it to the baseline BERTweet model. Apart from that, the procedure is similar to Step 2, adjusting the model with part of the examples and following a feature extraction procedure to generate and evaluate a classifier with the other part. The goal is to examine if the pruned model architecture suggested by the method generalizes well on a different data set. Because of that, we consider the datasets in G3.

For all training procedures, including those performed in Steps 1 and Step 2 of the method and in the generalization check, the language models were adjusted using AdamW (LOSHCHILOV; HUTTER, 2019), by setting the *learning rate* parameter to 2e-5, *adam epsilon* to 1e-8, *batch size* of 16. We ran for four *epochs*.

Lastly, to determine whether the differences among the results are statistically significant at a 0.05 significance level, we ran the t-test from the library SciPy (VIRTANEN et al., 2020). Whenever applicable, we present the results of the statistical tests in the results tables. We use the symbol $\succ$ to show that the results achieved by some model x are significantly better than those achieved by another model y so that $\{x\} \succ \{y\}$.

5.2 Experimental results

This section presents the experimental results of each step from the method and the generalization check. Section 5.2.1 presents the result of Step 1 (the order of importance of the heads) and three random orders of importance, which compose a random order strategy created to enrich the evaluation of our method. In Section 5.2.2 and Section 5.2.3, the results of Step 2 and the generalization check are respectively exhibited. Lastly, in Section 5.2.4, we explain the results achieved by running Step 2 using the three random orders of importance individually.

5.2.1 Results of Step 1 – Evaluating the importance order

At this point, one reasonable question is whether the order proposed by Step 1, referred to as OI, yields good results compared to simply choosing a random order of importance to the heads. Thus, besides comparing the results of the pruned models identified by our method to the complete model, we also compare them to pruned models detected after running Step 2 with a random order strategy. The result of the random order strategy is computed from the mean results of the execution of Step 2 with three different random orders of importance called A1, A2, and A3.

Figure 2 presents the order of importance established to the heads of the BERTweet model architecture using our proposed method and the random order strategy. The layers are enumerated from the bottom to the top of the model, i.e., layer 0 corresponds to the initial layer, while layer 11 is the model's final layer. The blue-toned heatmap refers to the output of Step 1 of our proposal, named here as OI. Each value in the cells represents the importance of that head, where the lower the value, the more important it is, making its chance of being removed lower. More clearly, the value 0 corresponds to the most important head (row 0, column 0, stronger blue), whereas the value 143 refers to the least important one (row 8, column 4, lighter blue), which would be the first to be removed during the pruning procedure.

The heatmaps in orange tones in Figure 2 show the three randomly generated orders of removal for the heads, namely A1, A2, and A3. As before, the stronger the color, the later the head is chosen to be removed.

As can be observed in the order of importance established in Step 1 (OI) presented in Figure 2, the most essential heads concentrate on the six bottom layers of the model and on the last one. It corroborates in parts the findings of Sajjad et al. (2023), which concluded that the bottom layers are most critical to keeping downstream task performance. On the other hand, the presence of so many important heads on the last layer also agrees with Jawahar, Sagot, and Seddah (2019), who stated that semantic features, likely important to sentiment analysis task, are in the top layers of BERT, closest to the classification layer.

5.2.2 Results of Step 2 – Evaluating the proposed approaches

As explained in Section 4.2, Step 2 can follow three different approaches: H1, H2, and H3. In this Subsection, we present the results achieved by each one of them when running with

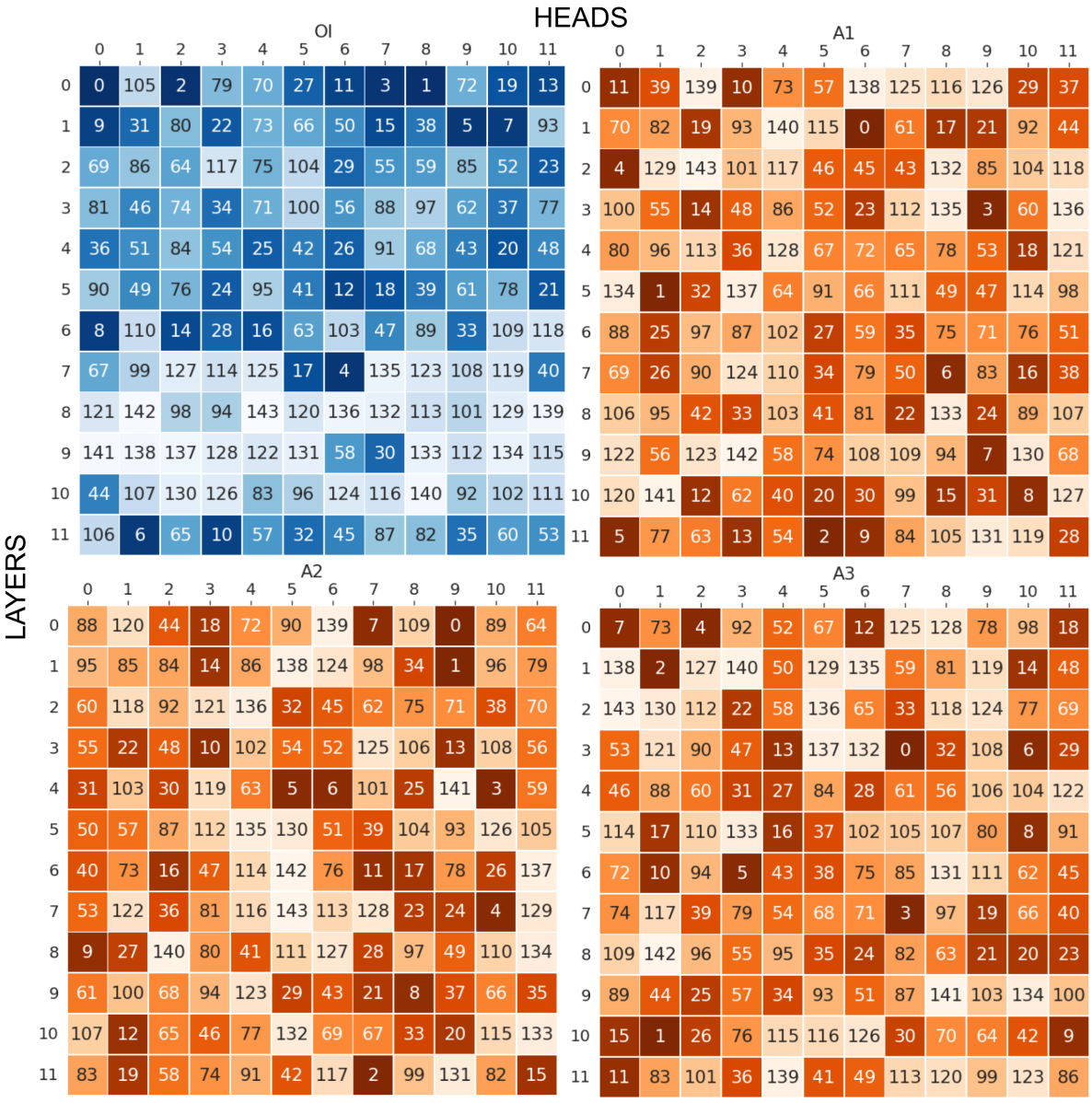

Figure 2: Importance order of the heads based on OI, the blue heatmap, and A1, A2 and A3, the orange heatmaps.

the order of importance established in Step 1 (OI). We compare them to the complete model (CM) results and the results when running the same approaches with the random order strategy.

The algorithm followed in H1 defines a ranking of 144 models (the complete and 143 pruned ones) just before deciding on the best-pruned model, as described in Subsection 4.2.1. Table 5 presents the top five models with their respective F1-macro mean scores for each dataset when running with the order of importance OI. As can be observed in this table, the model pruned by the first 68 heads was the best-ranked; thus, it was selected as the best. As explained in Section 4.2.1, the models' ranking procedure is calculated based on the sum of ranks each model reaches in each dataset. The lower the rank sum, the better the model. Even though the model pruned by the first 68 heads had achieved the best predictive performance only in one dataset (PERSON), its rank sum was the lower, reaching 188.0. The exact process was executed using the three random orders of importance: A1, A2, and A3. Table 6 summarizes the results of Step 2 when following the approach H1. Figure 3 exposes the architecture of the self-attention layers from each of these pruned models, which we refer to as MOIH1, MA1H1, MA2H1, and MA3H1 from this point on, where the first one stands for the model proposed by H1 using the order of importance established in Step 1. The others refer to the models proposed by H1 using the three random orders of importance.

Table 5: Top five models pruned according to H1, using OI. The lower the rank sum, the better the model. The results presented in this table are the F1 macro scores achieved by each pruned model using **G2**

Datasets	**Amount of heads pruned**				
	68	**61**	**69**	**72**	**65**
SemEval17	91.18	90.97	**91.25**	91.22	90.76
Vader	88.86	88.75	88.71	**88.90**	88.80
SentiStrength	85.56	**85.78**	85.52	85.51	85.66
SemEval18	86.96	86.96	86.86	86.97	**87.03**
movie	**72.50**	72.33	72.41	71.71	72.30
hobbit	74.67	**75.04**	74.89	74.43	75.01
person	70.54	**70.75**	70.66	70.36	70.59
Rank Sum	**188.0**	191.0	200.5	212.0	212.5
Rank Pos.	**1**	2	3	4	5

Before presenting the generalization performance of the pruned models generated by using H1 corresponding to the generalization check, we show its results using the datasets from (G2) in Table 7. In this table, we present the mean and standard deviation of the F1-score from all executions (10 seeds) of the complete model (CM), MOIH1, and the

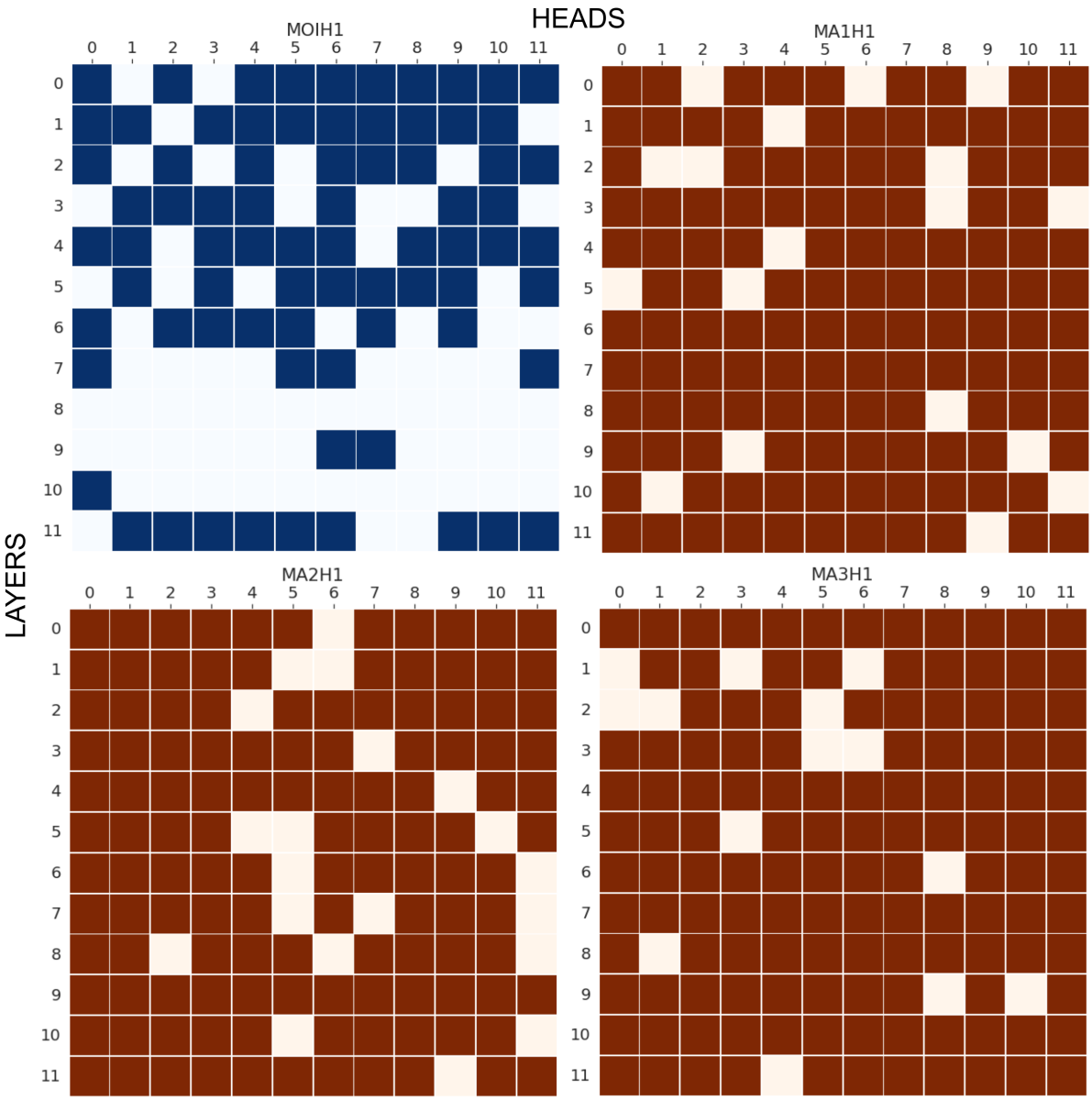

Figure 3: Architecture of the models (MOIH1, MA1H1, MA2H1, and MA3H1) after pruning heads according to the best amount of pruning achieved by H1 when using OI, A1, A2, and A3, respectively.

Table 6: Best amount of pruned heads based on OI, A1, A2 and A3 according to H1 achieved individually.

Order of importance	Best Amount of pruning
OI	68
A1	18
A2	20
A3	14

model resulting from the average results obtained by the pruned models using the three random orders of importance and H1, named MAMH1. The best results for each dataset are highlighted in the table. Moreover, the table presents a percentage comparison among CM and the other models to identify their performance's improvement or decline.

Table 7: Comparison among the results achieved by MAMH1, MOIH1, and CM over the Step 2 datasets (G2)

Datasets	F1-macro: Mean (STD)			t-test
	MAMH1	**MOIH1**	**CM**	
S17	91.08 (0.52) -0.18%	91.18 (0.40) -0.07%	**91.24 (0.43)** **100%**	-
vad	88.64 (0.58) +0.01%	**88.86 (0.52)** **+0.26%**	88.63 (0.84) 100%	-
SSt	85.86 (1.06) -0.19%	85.56 (1.05) -0.53%	**86.02 (0.74)** **100%**	-
S18	**87.57 (0.72)** **+0.54%**	86.96 (1.00) -0.16%	87.10 (0.66) 100%	-
mov	71.72 (2.08) +1.72%	**72.50 (2.38)** **+2.82%**	70.51 (3.64) 100%	-
hob	73.43 (2.97) +0.98%	**74.67 (3.89)** **+2.68%**	72.72 (3.33) 100%	-
per	69.52 (1.81) +1.12%	**70.54 (2.01)** **+2.60%**	68.75 (1.70) 100%	{MOIH1} $\succ$ {CM}

As can be observed in this table, MOIH1 yielded the best model in four datasets, whereas CM won in two and MAMH1 won in only one. When comparing exclusive MOIH1 to CM; the former triumphs in four datasets and the latter in the others three. The more expressive results of MOIH1 were achieved on the smaller datasets, and the result achieved in PER dataset was statistically significant at the level of significance established in our methodology. Curiously, PER is the dataset from G2 whose sentences have more tokens on average, 27.27 (standard deviation of 8.36), as exposed in Table 4. Comparing only the results achieved by MAMH1 to CM, we realize that MAMH1 won in five datasets, whereas CM won in two. However, none of these results was statistically significant at the level of

significance established in our methodology. Further, when comparing MOIH1 to MAMH1, the former overcomes the latter in five datasets, although without statistical significance.

As explained in subsections 4.2.2 and 4.2.3, during the execution of the approaches H2 and H3, the decision on the best-pruned model was based on a ranking of only seven pruned models in contrast to the 143 from H1. It happened because H2 and H3 generate one pruned model for each dataset, and we ran this step with seven datasets (G2). Different from H1, with which we can identify the generated pruned models by the number of heads pruned, in H2 and H3, the generated pruned models must be more detailed. Therefore, those generated by using OI with H2 and H3 are presented in Figures 4 and 5, respectively, with the ranked best highlighted in orange. Similarly to what we did with H1, H2 and H3 were also executed using the three random orders of importance. Figures 6 and 7 expose the architecture of the self-attention layers from each best-pruned model selected when using respectively H2 and H3 with the four orders of importance: OI, A1, A2, and A3. From this point on, we refer to these models as MOIH2, MA1H2, MA2H2, MA3H2, MOIH3, MA1H3, MA2H3, and MA3H3.

Analogous to what we did to H1, before presenting the generalization performance of the pruned models corresponding to the generalization check, we show the results achieved by H2 and H3 using the datasets from Step 2.

In Table 8, we present the mean and standard deviation of the F1-score from all executions (10 seeds) of the complete model (CM), MOIH2, and the model resulting from the average results obtained by the pruned models using the three random orders of importance and H2, named MAMH2. The best results for each dataset are highlighted in the table. Furthermore, the table presents a percentage comparison among CM and the other models to identify their performance's improvement or decline.

As can be observed in Table 8, MOIH2 yielded the best model in four datasets, whereas MAMH2 won in the other three. On the other hand, when comparing MOIH2 merely to CM, the former won in all datasets. Similar to MOIH1, the more expressive results of MOIH2 were achieved on the smaller datasets, and the result achieved in MOV dataset was statistically significant at the level of significance established in our methodology. Moreover, when comparing MAMH2 to CM, we realize that MAMH2 won in five datasets, whereas CM won in one, and we have a drawn on the S17 dataset. However, none of these results was statistically significant at the level of significance established in our methodology. Further, when comparing MOIH2 to MAMH2, the former overcome the latter in four datasets but without statistical significance.

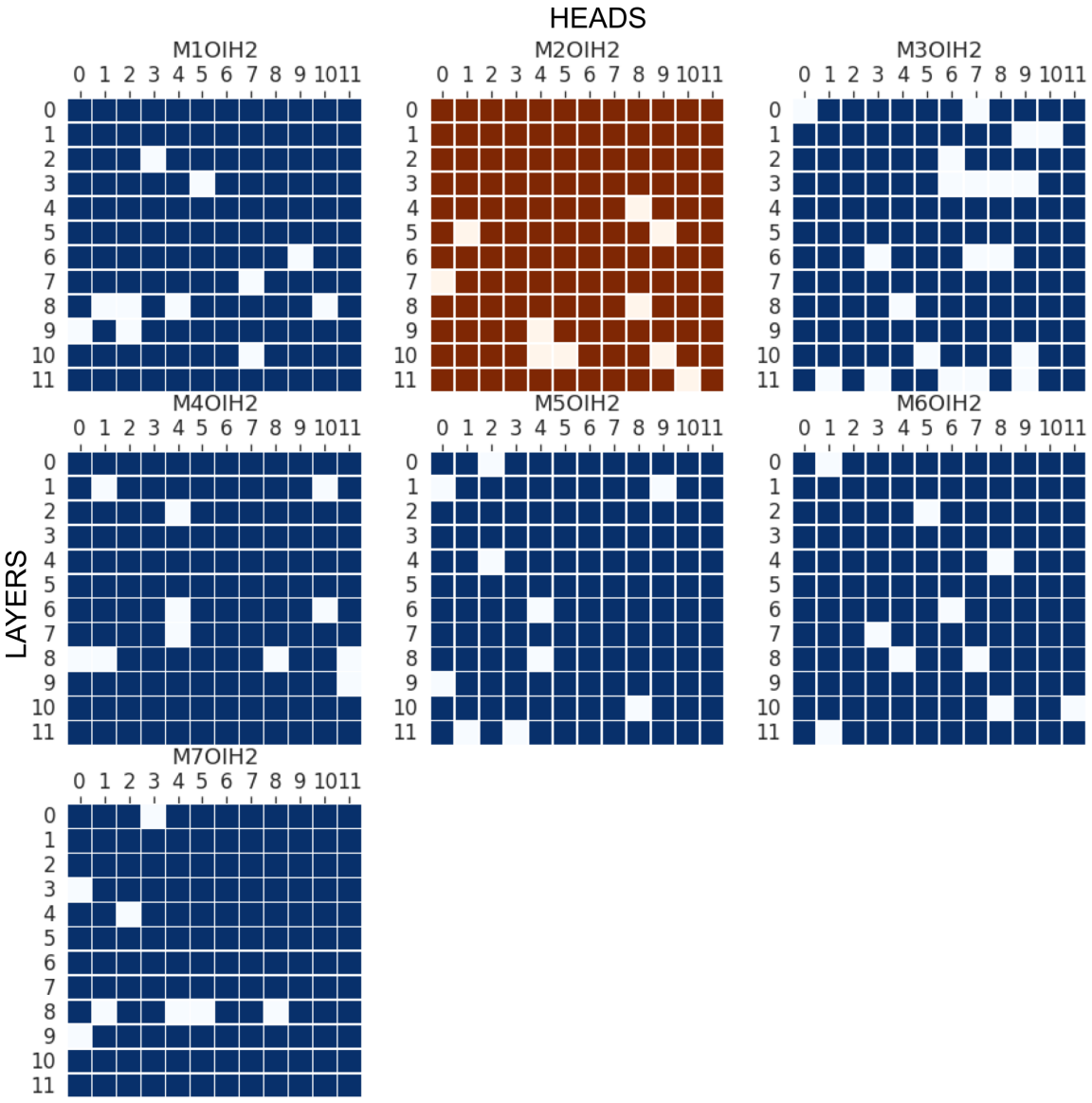

Figure 4: Architecture of the pruned models generated by approach H2 when using the orders of importance OI. The orange model is the ranked best.

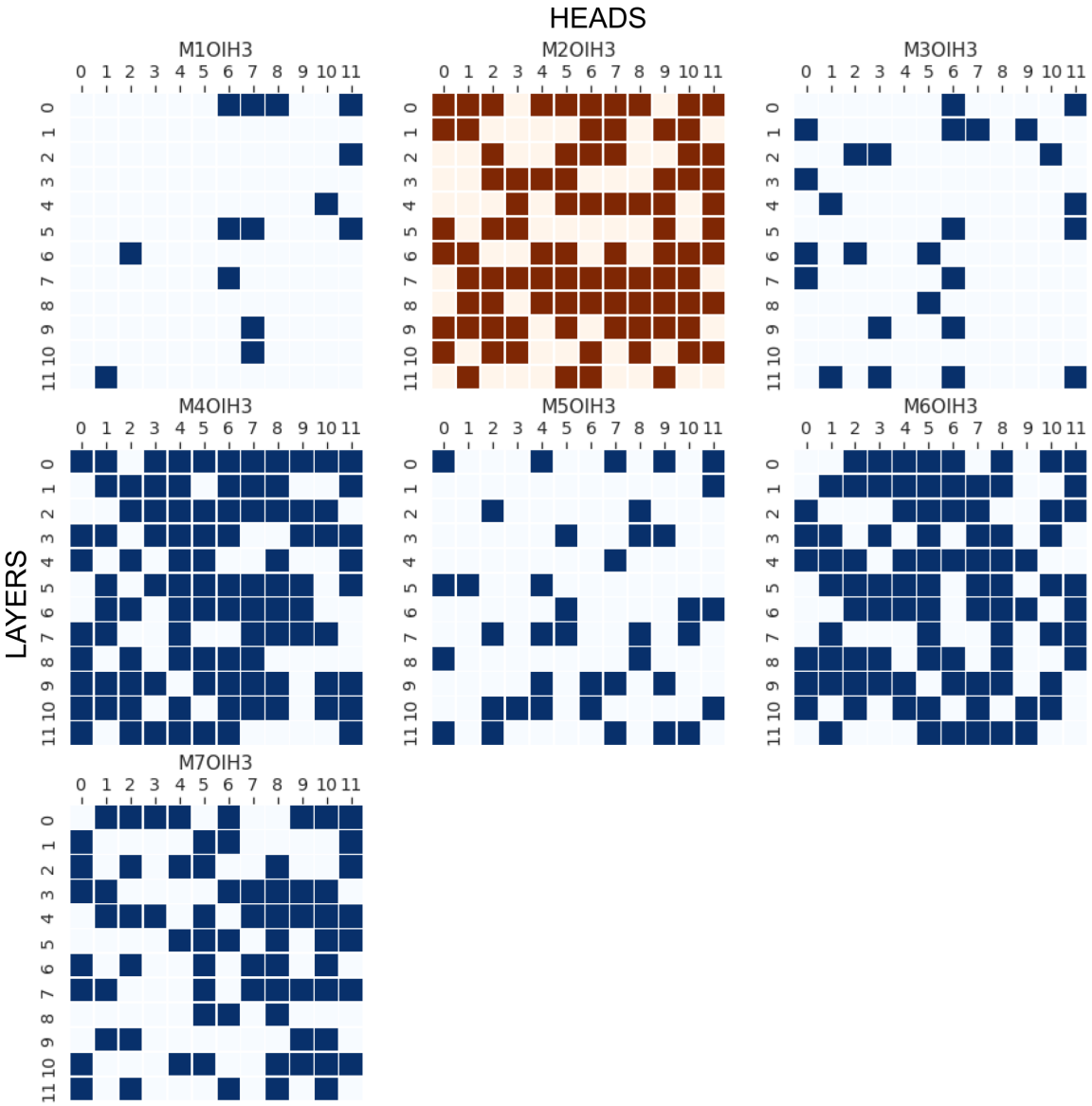

Figure 5: Architecture of the pruned models generated by the approach H3 when using the orders of importance OI. The orange model is the ranked best.

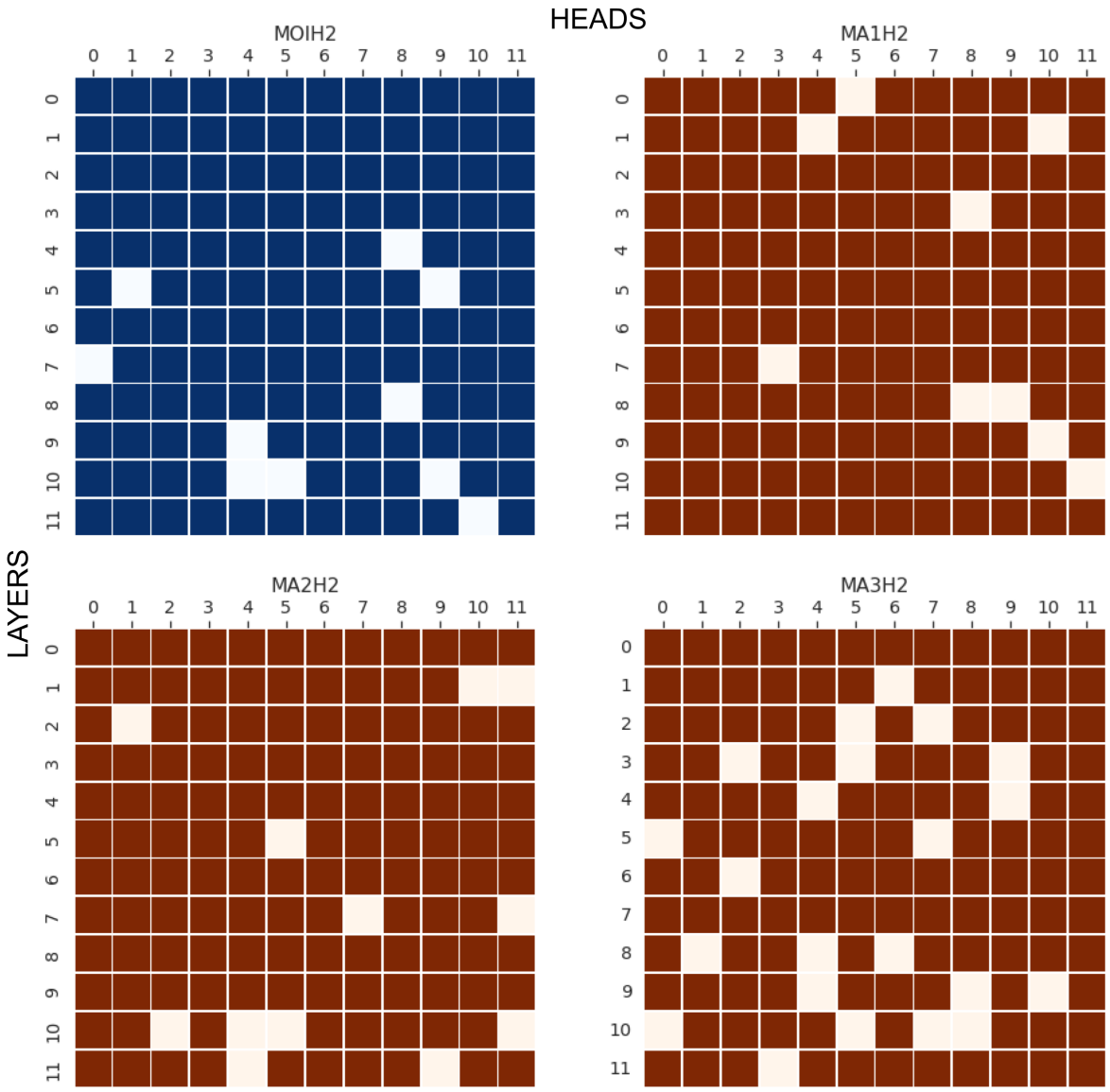

Figure 6: Architecture of the best pruned models (MOIH2, MA1H2, MA2H2, and MA3H2) identified by approach H2 when using the orders of importance OI, A1, A2, and A3 respectively.

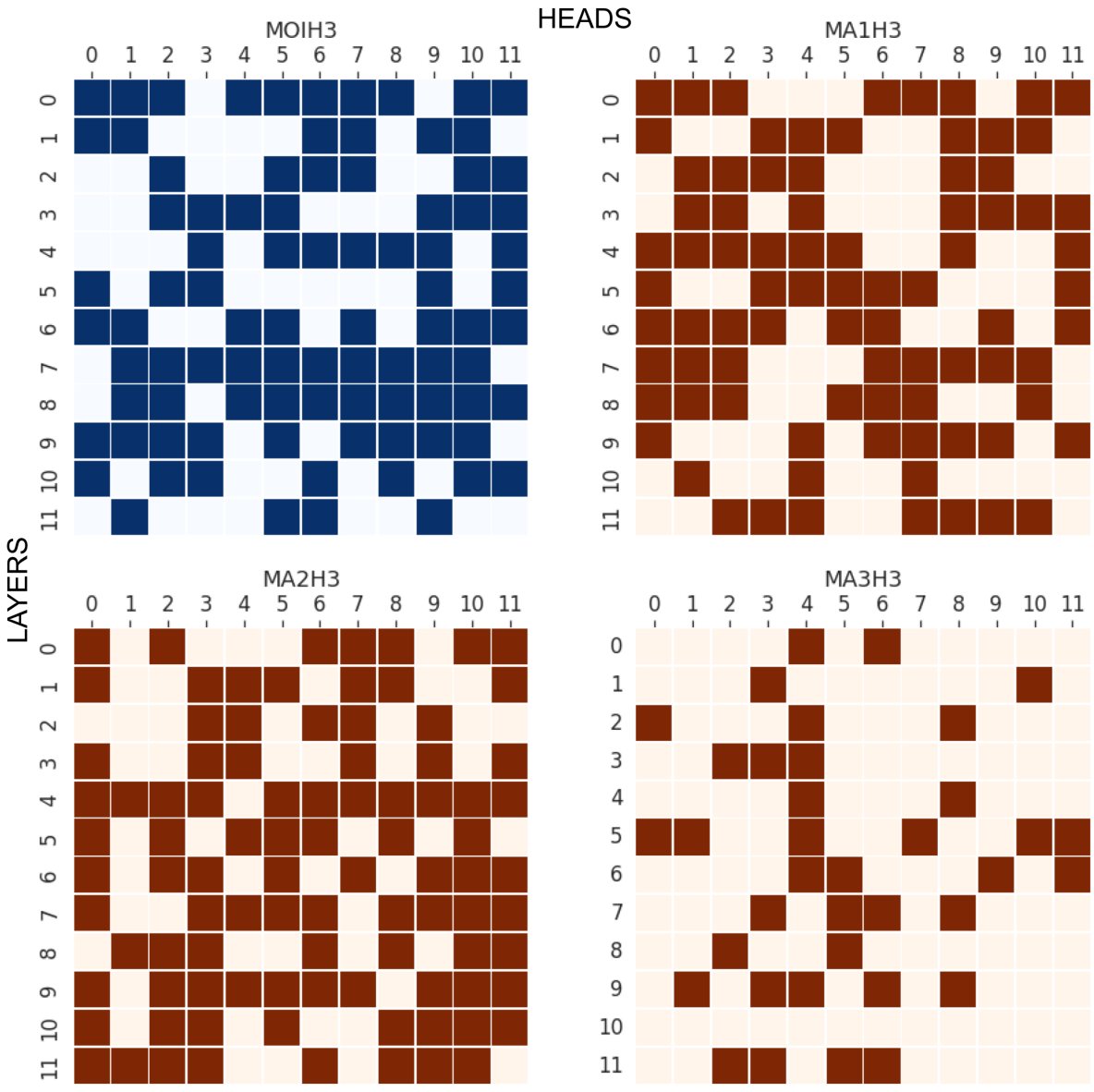

Figure 7: Architecture of the best pruned models (MOIH3, MA1H3, MA2H3, and MA3H3) identified by approach H3 when using the orders of importance OI, A1, A2, and A3 respectively.

Table 8: Comparison among the results achieved by MAMH2, MOIH2, and CM over the Step 2 datasets (G2)

Datasets	F1-macro: Mean (DTD)			t-test
	MAMH2	MOIH2	CM	
S17	91.24 (0.60) 0%	**91.26 (0.43)** **+0.02%**	91.24 (0.43) 100%	-
vad	**88.91 (0.82)** **+0.32%**	88.75 (0.56) +0.14%	88.63 (0.84) 100%	-
SSt	85.92 (0.81) -0.12%	**86.10 (0.88)** **+0.09%**	86.02 (0.74) 100%	-
S18	**87.82 (0.91)** **+0.83%**	87.71 (0.88) +0.70%	87.10 (0.66) 100%	-
mov	72.73 (1.98) +3.15%	**73.53 (1.67)** **+4.28%**	70.51 (3.64) 100%	{MOIH2} $\succ$ {CM}
hob	73.55 (2.98) +1.14%	**73.73 (2.46)** **+1.39%**	72.72 (3.33) 100%	-
per	**70.37 (2.10)** **+2.36%**	69.91 (2.16) +1.69%	68.75 (1.70) 100%	-

In Table 9, we present the mean and standard deviation of the F1-score from all executions (10 seeds) of the complete model (CM), MOIH3, and the model resulting from the average results obtained by the pruned models using the three random orders of importance and H3, named MAMH3. The best results for each dataset are highlighted in the table. Furthermore, the table presents a percentage comparison among CM and the other models to identify their performance's improvement or decline.

As can be observed in Table 9, MAMH3 yielded the best model in six datasets, whereas CM won in the other one. When comparing MOIH3 to CM exclusively, the former won in four datasets, but with statistically significant only in S18, the latter won in the other three with statistical significance in S17. Moreover, when comparing MAMH3 to CM, we realize that MAMH3 won in six datasets with statistical significance in S18 and PER datasets, whereas CM won in one. Further, when comparing MOIH3 to MAMH3, the latter overcomes the former in all datasets but without statistical significance.

In Table 10, we bring all these results together to directly compare them. We can observe in this table that MAMH3 yielded the best model in four datasets, whereas MOIH2 won in three. When we compare the pruned models identified by our method to the complete one, we realize that all three approaches achieved very good predictive performance. In some cases, these results were statistically significant at the level of significance established in our methodology. More expressive results were achieved on smaller datasets,

Table 9: Comparison among the results achieved by MAMH3, MOIH3, and CM over the Step 2 datasets (G2)

Datasets	F1-macro: Mean (STD)			t-test
	MAMH3	MOIH3	CM	
S17	91.05 (0.52) -0.21%	90.69 (0.46) -0.60%	**91.24 (0.43)** **100%**	{CM} ≻ {MOIH3}
vad	**88.92 (0.62)** **+0.33%**	88.65 (1.64) +0.02%	88.63 (0.84) 100%	-
SSt	**86.06 (0.77)** **+0.05%**	85.48 (0.87) -0.63%	86.02 (0.74) 100%	-
S18	**89.03 (0.53)** **+2.22%**	87.85 (0.73) +0.86%	87.10 (0.66) 100%	{MOIH3} ≻ {CM} {MAMH3} ≻ {CM}
mov	**70.90 (3.17)** **+0.55%**	70.74 (3.08) +0.33%	70.51 (3.64) 100%	-
hob	**74.81 (1.89)** **+2.87%**	73.68 (2.49) +1.32%	72.72 (3.33) 100%	-
per	**71.99 (0.93)** **+4.71%**	68.07 (2.16) -0.99%	68.75 (1.70) 100%	{MAMH3} ≻ {CM}

indicating that, for these cases, a huge pre-trained model has too much superfluous information that harms the finetuning quality applied.

If for one side, it is crucial to evaluate the pruned models in terms of predictive performance, for the other side, we are interested in identifying how compressed they are. Table 11 presents the number of pruned heads from the best-pruned models determined by our method and by each of the models that compound the random strategy.

Notably, the MOIH1 is the most compressed model identified by our method, although MOIH3 is very compact as well. Regardless of being so compacted, we can observe that they achieved an outstanding predictive performance. MOIH1 did not win in any dataset, but, with statistical significance, its results were only overcome in dataset S18 by MOIH3 and MAMH3; however, in dataset PER, its results were better than MOIH3 and in dataset S17, better than CM and MOIH3. MOIH3, on the other hand, was overcome with statistical significance in four datasets, but in only one by the complete model CM.

Surprisingly, the random order strategy MAMH3, which is composed of very compressed models, achieved a spectacular predictive performance compared to the complete model. Compared to all models, it wins in four datasets, but compared to only the complete model, this model wins in six. In Section 5.2.4, we detail the results achieved by the random order strategy applied to the datasets from the generalization check.

Table 10: Comparison among the results achieved by MAMH1, MOIH1, MAMH2, MOIH2, MAMH3, MOIH3 and CM over the Step 2 datasets (G2)

Datasets	F1-macro: Mean (STD)							t-test
	MAMH1	MOIH1	MAMH2	MOIH2	MAMH3	MOIH3	CM	
S17	91.08 (0.52) -0.18%	91.18 (0.40) -0.07%	91.24 (0.60) 0%	**91.26** **(0.43)** **+0.02%**	91.05 (0.52) -0.21%	90.69 (0.46) -0.60%	91.24 (0.43) 100%	{CM} ≻ {MOIH3} {MOIH1} ≻ {MOIH3} {MOIH2} ≻ {MOIH3} {MAMH2} ≻ {MOIH3}
vad	88.64 (0.58) +0.01%	88.86 (0.52) +0.26%	88.91 (0.82) +0.32%	88.75 (0.56) +0.14%	**88.92** **(0.62)** **+0.33%**	88.65 (1.64) +0.02%	88.63 (0.84) 100%	-
SSt	85.86 (1.06) -0.19%	85.56 (1.05) -0.53%	85.92 (0.81) -0.12%	**86.10** **(0.88)** **+0.09%**	86.06 (0.77) +0.05%	85.48 (0.87) -0.63%	86.02 (0.74) 100%	-
S18	87.57 (0.72) +0.54%	86.96 (1.00) -0.16%	87.82 (0.91) +0.83%	87.71 (0.88) +0.70%	**89.03** **(0.53)** **+2.22%**	87.85 (0.73) +0.86%	87.10 (0.66) 100%	{MOIH3} ≻ {CM} {MOIH3} ≻ {MOIH1} {MAMH3} ≻ {CM} {MAMH3} ≻ {MOIH1} {MAMH3} ≻ {MOIH2} {MAMH3} ≻ {MOIH3}
mov	71.72 (2.08) +1.72%	72.50 (2.38) +2.82%	72.73 (1.98) +3.15%	**73.53** **(1.67)** **+4.28%**	70.90 (3.17) +0.55%	70.74 (3.08) +0.33%	70.51 (3.64) 100%	{MOIH2} ≻ {CM} {MOIH2} ≻ {MAMH3} {MOIH2} ≻ {MAMH1} {MOIH2} ≻ {MOIH3}
hob	73.43 (2.97) +0.98%	74.67 (3.89) +2.68%	73.55 (2.98) +1.14%	73.73 (2.46) +1.39%	**74.81** **(1.89)** **+2.87%**	73.68 (2.49) +1.32%	72.72 (3.33) 100%	-
per	69.52 (1.81) +1.12%	70.54 (2.01) +2.60%	70.37 (2.10) +2.36%	69.91 (2.16) +1.69%	**71.99** **(0.93)** **+4.71%**	68.07 (2.16) -0.99%	68.75 (1.70) 100%	{MOIH1} ≻ {CM} {MOIH1} ≻ {MOIH3} {MAMH2} ≻ {MOIH3} {MAMH3} ≻ {CM} {MAMH3} ≻ {MOIH2} {MAMH3} ≻ {MOIH3}

The results achieved by MOIH1, MOIH2, and MOIH3 indicate that we can positively answer the research question RQ1 once these models present similar or even better predictive performance than the complete one. Moreover, MOIH1 and MOIH3 are significantly compressed, indicating that we can positively answer the research question RQ2. However, to reinforce those answers, we demonstrate the generalization of these results to different datasets in Section 5.2.3.

5.2.3 Generalization Check – Evaluating the generalization ability

After executing the generalization check, the results aim to verify the pruned models' generalization capabilities. Table 12 presents the mean and standard deviation of the F1-score from all executions (10 seeds) of CM, MOIH1, MAMH1, MOIH2, MAMH2, MOIH3, and MAMH3, over the generalization check's datasets (G3). The best results for each dataset

Table 11: Comparison among the number of pruned heads from MOIH1, MA1H1, MA2H1, MA3H1, MOIH2, MA1H2, MA2H2, MA3H2, MOIH3, MA1H3, MA2H3, and MA3H3

Model	Number of pruned heads
MA3H3	107 (74.31%)
MOIH1	**68 (47.22%)**
MA1H3	61 (42.36%)
MOIH3	**55 (38.19%)**
MA2H3	51 (35.42%)
MA3H2	22 (15.28%)
MA2H1	20 (13.89%)
MA1H1	18 (12.50%)
MA3H1	14 (9.72%)
MA2H2	12 (8.33%)
MOIH2	**10 (6.94%)**
MA1H2	09 (6.25%)

are highlighted in the table. Moreover, the table presents a percentage comparison among CM and all the other models to identify their performance's improvement or decline.

As can be observed in Table 12, MAMH3 won in six datasets, whereas MOIH1 won in the other two. Similar to what we observed in Step 2, the most relevant improvements were on the smaller datasets.

Comparing the pruned models detected by our method to the complete one, it is possible to realize that MOIH1 won in two datasets (STM and AIS) with statistical significance and won in four more datasets without it. However, MOIH1 lost in the other two datasets, but without statistical significance. MOIH2 overcame CM in all datasets, although without presenting statistical significance. MOIH3 won in two datasets, but we recognized that it had the worst predictive performance among them, being surpassed by CM in six datasets, albeit in just one (OMD), with statistical significance.

On the other hand, when we compare the results of MOIH1 to MOIH2, MOIH1 surpassed MOIH2 in half of the datasets, while being the smallest identified by our method as exposed in Table 13. With statistical significance, MOIH2 overcame MOIH1 in one dataset (SAN) whereas MOIH1 won MOIH2 in another one (AIS).

When we compare MOIH1 to the pruned models identified by the random strategy (MAMH1, MAMH2, and MAMH3), a highlight must be done to MAMH3. MAMH3 is composed of very compressed models and won in six datasets while losing to MOIH1 in the other two. Compared to CM, MAMH3 won in all datasets. Such a good performance was also observed in Step 2. MAMH1 and MAMH2, in turn, are not so pruned. Although they

Table 12: Comparison among the results (F1 Mean and STD, with a percentage comparison regarding CM) achieved by MAMH1, MOIH1, MAMH2, MOIH2, MAMH3, MOIH3 and CM, over the generalization check datasets (G3).

Datasets	F1-macro: Mean (STD)							t-test
	MAMH1	MOIH1	MAMH2	MOIH2	MAMH3	MOIH3	CM	
S13	84.14 (2.39) -0.50%	84.58 (1.78) +0.02%	84.61 (1.76) +0.06%	84.70 (1.55) +0.17%	**85.01** **(1.31)** **+0.53%**	84.95 (1.66) +0.46%	84.56 (1.73) 100%	-
Tar	84.45 (0.80) +0.27%	84.27 (0.77) +0.06%	84.62 (0.80) +0.47%	84.62 (0.75) +0.47%	**85.06** **(0.78)** **+1.00%**	83.98 (0.99) -0.29%	84.22 (0.75) 100%	{MAMH3} ≻ {CM} {MAMH3} ≻ {MOIH3} {MAMH3} ≻ {MOIH1}
OMD	83.47 (0.99) +0.01%	82.84 (1.20) -0.74%	83.29 (0.93) -0.20%	83.49 (1.66) +0.04%	**83.73** **(1.01)** **+0.32%**	81.66 (0.99) -2.16%	83.46 (1.13) 100%	{CM} ≻ {MOIH3} {MOIH1} ≻ {MOIH3} {MOIH2} ≻ {MOIH3} {MAMH1} ≻ {MOIH3} {MAMH2} ≻ {MOIH3} {MAMH3} ≻ {MOIH3}
Nar	93.05 (1.30) +0.14%	93.28 (1.43) +0.39%	92.97 (1.33) +0.05%	93.24 (1.22) +0.34%	**94.01** **(0.72)** **+1.17%**	92.88 (1.15) -0.04%	92.92 (1.44) 100%	{MAMH3} ≻ {MOIH3}
san	85.68 (1.47) -0.09%	84.96 (1.34) -0.93%	85.74 (1.36) -0.02%	86.36 (1.46) +0.70%	**86.92** **(0.91)** **+1.35%**	84.56 (1.00) -1.39%	85.76 (1.96) 100%	{MAMH3} ≻ {MOIH1} {MAMH3} ≻ {MOIH3} {MOIH2} ≻ {MOIH1} {MOIH2} ≻ {MOIH3} {MAMH2} ≻ {MOIH3}
iph	74.09 (2.59) +0.38%	75.16 (2.23) +1.83%	74.61 (2.49) +1.08%	75.06 (2.53) +1.69%	**75.67** **(1.98)** **+2.52%**	72.97 (2.16) -1.14%	73.81 (2.49) 100%	{MOIH1} ≻ {MOIH3} {MAMH3} ≻ {MOIH3}
stm	78.82 (3.78) +0.33%	**82.40** **(2.89)** **+4.89%**	79.70 (3.22) +1.45%	81.11 (2.98) +3.25%	82.00 (2.98) +4.38%	76.68 (3.54) -2.39%	78.56 (4.29) 100%	{MOIH1} ≻ {CM} {MOIH1} ≻ {MAMH1} {MOIH1} ≻ {MOIH3} {MOIH2} ≻ {MOIH3} {MAMH3} ≻ {MOIH3} {MAMH3} ≻ {CM}
ais	82.25 (3.13) +0.60%	**86.68** **(3.29)** **+6.02%**	82.77 (3.34) +1.24%	82.20 (3.74) +0.54%	84.90 (2.44) +3.84%	84.14 (3.29) +2.91%	81.76 (4.08) 100%	{MAMH3} ≻ {CM} {MOIH1} ≻ {CM} {MOIH1} ≻ {MAMH1} {MOIH1} ≻ {MAMH2} {MOIH1} ≻ {MOIH2}

have not won in any dataset, they achieved competitive performance in six and seven, respectively. With statistical significance, MOIH1 won MAMH1 in the dataset STM and won MAMH1 and MAMH2 in the dataset AIS. In Section 5.2.4, we detail the results achieved by the random order strategy applied to the datasets from the generalization check. The reasons for the excellent performance of MAMH3 are discussed there.

These results enhance the answers to the two research questions RQ1 and RQ2. We uncover pruned models which perform as well as the complete model, or even better, in datasets that the method did not have access to during the pruning strategy. Furthermore, our method proved efficient in identifying a significantly pruned model, MOIH1.

Table 13: Number of parameters and allocated disk space by CM, and the pruned models

Model	Allocated Disk Space	Number of Parameters
CM	539,950,787 MB (100.00%)	134,965,505.00 (100.00%)
MOIH1	468,297,089 MB (-13.27%)	117,053,633 (-13.27%)
MOIH2	532,079,023 MB (-1.46%)	132,997,505 (-1.46%)
MOIH3	496,655,519 MB (-8.02%)	124,141,505 (-8.02%)
MA1H1	525,781,548 MB (-2.62%)	131,423,105 (-2.62%)
MA2H1	524,207,185 MB (-2.92%)	131,029,505 (-2.92%)
MA3H1	528,930,275 MB (-2.04%)	132,210,305 (-2.04%)
MA1H2	532,866,215 MB (-1.31%)	133,194,305 (-1.31%)
MA2H2	530,504,627 MB (-1.75%)	132,603,905 (-1.75%)
MA3H2	522,632,782 MB (-3.21%)	130635905 (-3.21%)
MA1H3	491,932,378 MB (-8.89%)	122,960,705 (-8.89%)
MA2H3	499,804,282 MB (-7.44%)	124,928,705 (-7.44%)
MA3H3	436,809,392 MB (-19.10%)	109,181,633 (-19.10%)

In addition, we have also recorded the time spent to finetune the complete model CM and the pruned model MOIH1 – the most pruned model proposed by our method that still achieves a good predictive performance – for all executions (10 seeds). Table 14 summarizes these results, presenting the mean time spent for each dataset. We can realize that MOIH1 provokes improvements of more than 20% when compared to the complete model.

5.2.4 Individual results from the random orders of importance

So far, we have only exposed the predictive performance resulting from the random order strategy without individually presenting the results from the pruned models which compose it. In this section, we provide further details on these individual results once a random choice selected by chance might perform well (CHEN et al., 2020).

Table 15 presents the mean F1-macro results achieved by the models pruned with our approaches (MOIH1, MOIH2, and MOIH3), the complete model (CM), and the pruned models based on the random orders of importance (MA1H1, MA2H1, MA3H1, MA1H2, MA2H2, MA3H2, MA1H3, MA2H3, MA3H3), over the generalization check datasets (G3). When evaluating the models generated by pruning according to random orders of importance separately, we observed that these pruned models could provide as good results as the complete model or even better.

Remarkably, the model MA3H3 presents an astonishing predictive performance and still is the most compressed model among all, as observed in Table 13. For most of the

Table 14: Mean time, in seconds, spent to finetune the complete model CM, the pruned model MOIH1 and the pruned model MA3H3. The table presents the proportion in relation to the complete model and the standard deviation (in parentheses).

Dataset	Mean time spent to finetune		
	MA3H3	**MOIH1**	**CM**
SemEval13	61.81 (10.23) -55.07%	114.57 (19.53) -16.72%	137.58 (21.15) 100%
Target-dependent	46.10 (3.58) -55.83%	91.08 (7.33) -12.74%	104.38 (8.94) 100%
OMD	20.61 (1.02) -55.94%	40.16 (2.99) -14.15%	46.78 (4.29) 100%
Narr	14.07 (0.53) -58.06%	26.47 (1.56) -21.10%	33.55 (3.30) 100%
sanders	15.26 (0.56) -57.45%	29.57 (0.68) -17.54%	35.86 (4.51) 100%
iphone6	6.52 (0.11) -53.13%	12.42 (0.30) -10.71%	13.91 (0.62) 100%
sentiment140	5.39 (0.24) -56.64%	10.18 (0.66) -18.10%	12.43 (2.03) 100%
aisopos	4.11 (0.06) -52.81%	7.64 (0.14) -12.28%	8.71 (0.90) 100%

datasets, its results surpassed all the other models with statistical significance, and for the dataset STM, it promoted almost 12% of improvement. Once this pruned model was generated from a random order of importance, we cannot unhesitatingly determine what induced such good performance; however, observing its architecture, we realize that ten of the twelve layers have, in the maximum, four heads. It indicates that we can keep just a few heads for each layer from the original model, to perform the finetuning procedure, analogous to the finding of Michel, Levy, and Neubig (2019), which identified that for most layers, one head is indeed sufficient at inference time.

Although the approach H3 generates the best-pruned model (MA3H3) with a random order of importance, when using it with the proposed order of importance OI, the produced model did not yield so good results. MOIH3 presented a decline in its performance in six datasets compared to the complete model. However, when applying the approach H1 with OI, the model MOIH1 reaches outstanding performance, mainly in the small datasets. It indicates that the definition of a presumably favorable order of importance is not always aligned with how the approaches perform the pruning process.

Considering the compression, MA1H3 and MA2H3 have also achieved good performance compared to the complete model; however, they did not stand out in any dataset as much as MOIH1 and MA3H3 regarding the predictive performance.

In addition, we have also recorded the time spent to finetune the MA3H3 model when using the generalization check's datasets (G3). Table 14 presents the mean time spent in each of these datasets. We can realize that MA3H3 provokes improvements of more than 58% when compared to the complete model; however, it is essential to consider that the hardware used to compute this time was different from that used in the complete model, as explained in Section 5.1.

The results achieved by MA1H3, MA2H3, and MA3H3 corroborate to answer our research questions RQ1 and RQ2 positively once we demonstrate that even when applying a random order of importance of the heads, the approach H3 yield significantly pruned models that reach as good results as the complete one or even better.

Table 15: Results (F1 Mean and STD, with a percentage comparison regarding CM) achieved by the complete model (CM)) and all the best-pruned models identified in Step 2 over the generalization check datasets (G3).

Models	**F1-macro: Mean (STD)**							
	S13	**Tar**	**OMD**	**Nar**	**san**	**iph**	**stm**	**ais**
CM	84.56 (1.73) 100.0%	84.22 (0.75) 100.0%	83.46 (1.13) 100.0%	92.92 (1.44) 100.0%	85.76 (1.96) 100.0%	73.81 (2.49) 100.0%	78.56 (4.29) 100.0%	81.76 (4.08) 100.0%
MA1H1	84.85 (1.99) +0.34%	84.66 (0.86) +0.52%	83.89 (0.84) +0.52%	93.36 (1.33) +0.47%	85.76 (2.23) 0.00%	74.56 (2.92) +1.02%	78.77 (4.87) +0.27%	81.82 (3.86) +0.07%
MA2H1	83.90 (3.28) -0.78%	84.19 (1.07) -0.04%	83.10 (1.48) -0.43%	92.77 (1.18) -0.16%	85.56 (1.48) -0.23%	73.36 (2.67) -0.61%	78.21 (4.08) -0.45%	82.06 (2.86) +0.37%
MA3H1	83.67 (3.57) -1.05%	84.51 (0.92) +0.34%	83.42 (1.26) -0.05%	93.02 (1.65) +0.11%	85.71 (1.25) -0.06%	74.34 (2.40) +0.72%	79.49 (3.80) +1.18%	82.88 (3.65) +1.37%
MOIH1	84.58 (1.78) +0.02%	84.27 (0.77) +0.06%	82.84 (1.20) -0.74%	93.28 (1.43) +0.39%	84.96 (1.34) -0.93%	75.16 (2.23) +1.83%	82.40 (2.89) +4.89%	86.68 (3.29) +6.02%
MA1H2	84.82 (1.95) +0.31%	84.58 (0.84) +0.43%	83.47 (1.03) +0.01%	92.98 (1.39) +0.06%	86.15 (1.62) +0.45%	74.82 (2.34) +1.37%	79.04 (4.17) +0.61%	83.26 (3.93) +1.83%
MA2H2	84.63 (1.58) +0.08%	84.68 (0.84) +0.55%	84.12 (1.33) +0.79%	93.30 (1.67) +0.41%	85.75 (1.56) -0.01%	74.75 (2.65) +1.27%	79.73 (2.63) +1.49%	82.62 (2.97) +1.05%
MA3H2	84.39 (1.84) -0.20%	84.61 (0.84) +0.46%	82.29 (0.96) -1.40%	92.62 (1.21) -0.32%	85.31 (1.09) -0.52%	74.26 (2.71) +0.61%	80.30 (3.26) +2.21%	82.43 (3.78) +0.82%
MOIH2	84.70 (1.55) +0.17%	84.62 (0.75) +0.47%	83.49 (1.66) +0.04%	93.24 (1.22) +0.34%	86.36 (1.46) +0.70%	75.06 (2.53) +1.69%	81.11 (2.98) +3.25%	82.20 (3.74) +0.54%
MA1H3	85.02 (1.68) +0.54%	84.76 (0.97) +0.64%	82.90 (1.52) -0.67%	93.24 (0.85) +0.34%	86.09 (1.18) +0.38%	74.40 (2.15) +0.80%	79.63 (4.88) +1.36%	82.33 (2.42) +0.7%
MA2H3	84.47 (1.69) -0.11%	84.37 (0.73) +0.18%	84.03 (0.88) +0.68%	93.50 (1.13) +0.62%	85.21 (1.61) -0.64%	75.12 (2.34) +1.77%	78.75 (4.09) +0.24%	84.60 (4.21) +3.47%
MA3H3	**85.55 (0.98)** **+1.17%**	**86.04 (0.88)** **+2.16%**	**84.25 (1.16)** **+0.95%**	**95.29 (0.85)** **+2.55%**	**89.45 (0.92)** **+4.30%**	**77.50 (2.12)** **+5.00%**	**87.63 (1.86)** **+11.55%**	**87.78 (2.61)** **+7.36%**
MOIH3	84.95 (1.66) +0.46%	83.98 (0.99) -0.28%	81.66 (0.99) -2.16%	92.88 (1.15) -0.04%	84.56 (1.00) -1.40%	72.97 (2.16) -1.14%	76.68 (3.54) -2.39%	84.14 (3.29) +2.91%
t-test		{MA3H3} ≻ {ALL}	{CM} ≻ {MOIH3, MA3H2} {MOIH1} ≻ {MOIH3} {MOIH2} ≻ {MOIH3} {MA1H1} ≻ {MOIH1, MOIH3, MA3H2} {MA2H1} ≻ {MA3H2, MOIH3} {MA3H1} ≻ {MA3H2, MOIH3} {MA1H2} ≻ {MOIH3, MA3H2} {MA2H2} ≻ {MOIH1, MOIH3, MA3H2} {MA1H3} ≻ {MOIH3} {MA2H3} ≻ {MOIH3, MOIH1, MA3H2} {MA3H3} ≻ {MOIH1, MOIH3, MA1H3, MA3H2}	{MA3H3} ≻ {ALL}	{MA3H3} ≻ {ALL} {MA1H3} ≻ {MOIH3} {MA1H2} ≻ {MOIH3} {MA3H1} ≻ {MOIH3} {MOIH2} ≻ {MOIH3, MOIH1}	{MOIH1} ≻ {MOIH3} {MA2H3} ≻ {MOIH3} {MA3H3} ≻ {ALL}	{MOIH1} ≻ {CM, MOIH3, MA2H1, MA1H2, MA2H2, MA2H3} {MOIH2} ≻ {MOIH3} {MA2H2} ≻ {MOIH3} {MA3H2} ≻ {MOIH3} {MA3H3} ≻ {ALL}	{MOIH1} ≻ {CM, MOIH2, MA1H1, MA2H1, MA3H1, MA1H2, MA2H2, MA3H2, MA1H3} {MA3H3} ≻ {CM, MOIH2, MOIH3, MA1H3, MA1H1, MA2H1, MA3H1, MA1H2, MA2H2, MA3H2}

6 Conclusion and Future Works

While deeper and wider Transformer-based models have achieved remarkable performance in several NLP tasks, they also require substantial computational resources, limiting their usability and increasing carbon emission. Moreover, it sounds intuitive that relying on such huge models to solve NLP tasks applied to short texts, like tweets, may be excessive.

Traditionally, a common framework adopted to train Transformer-based models is to finetune a pre-trained model to a specific task. Michel, Levy, and Neubig (2019) demonstrated that it is beneficial to adopt one additional step, namely, pruning portions of finetuned models to be used at inference time, reducing the amount of computational resources to the inference procedure while achieving similar or even better performance. However, performing the finetuning procedure over the original huge model would still demand substantial computational resources, mainly when it is necessary to finetune the model several times or make several copies on the disk.

This dissertation focused on removing BERTweet heads before finetuning the model for the tweets sentiment analysis task. BERTweet is a BERT-based model trained with tweets. Our intuition is that the several heads and layers of BERT-based models retain redundant information that could be safely discarded. This way, the proposed pruned model would become more compatible with simpler hardware, requesting fewer computational resources and contributing to less carbon emission.

We proposed two research questions to guide our investigation:

RQ1 : Are there finetuned BERTweet pruned models that achieve competitive or even better predictive performance than the finetuned complete model in the tweets sentiment analysis task?

RQ2 : If so, how could one discover a significantly pruned model that reaches such performance?

We positively answered these questions by creating a method that prunes heads and layers from the original model by identifying the most relevant heads according to the gradients for the sentiment analysis task. We experimented with removing the heads incrementally, starting from the less to the most important ones. Using this paradigm, we implemented three approaches (H1, H2, and H3) so that pruned models were generated and evaluated against the sentiment analysis task. Based on the predictive performance of the pruned models, we selected the best of each approach and compared them to each other and the complete model, besides comparing them to pruned models identified by a random order strategy. The selected pruned models from the approach H1 are significantly pruned and yield competitive or even better performance than the complete one. Surprisingly, the three pruned models generated by approach H3 using the random orders of importance are significantly compact and achieved competitive or even better performance than the complete one. A highlight must be done to the pruned model resulting from the approach H3 and random order of importance A3, which is the most pruned and reaches the best predictive performance among all.

This way, the experimental results pointed out that there are sub-networks within BERTweet architecture that can be finetuned and applied to the sentiment analysis of tweets. They perform as well or even better than doing the same process with the complete BERTweet model while still saving computational resources. The approach H1 using the order of importance OI, proposed in this dissertation, pruned 47.22% of BERTweet heads (68 from 144 heads), corresponding to a reduction of 13.27% of the physical model. This reduction has also saved at least 10% of the time spent adapting the language model parameters while achieving the same or better predictive performance than the original network with a significance level of 0.05. On the other hand, the approach H3 using the random order of importance A3, pruned 74.31% of the BERTweet heads (107 from 144 heads), corresponding to a reduction of 19.19% of the physical model. This reduction has also saved at least 52% of the time spent adapting the language model parameters while achieving better predictive performance than the complete model. The results we achieved are evidence of the findings observed by Frankle and Carbin (2019) and corroborated by Prasanna, Rogers, and Rumshisky (2020), and Chen et al. (2020), once they reported that it is possible to uncover sub-networks that perform as good as a larger counterpart neural model or even better.

Therefore, we advise adding the pruning step before the finetuning as part of the traditional framework, at least when using it for tweets sentiment analysis. Notwithstanding, although we have implemented and evaluated our proposed method based on BERTweet

and the sentiment analysis task, we believe it is generic enough to be used for other tasks or Transformer-based models.

This study has some limitations. The first limitation is that we didn't compare the results achieved by the pruned models selected by our method to small models available and pruned models generated using other pruning strategies. The second limitation is the missing evaluation in Step 2 using the same finetuning procedure and classifier adopted in Step 1. It seems more coherent, but we could not experiment with it due to limited hardware resources. Another limitation concerns the lack of a deep quality analysis regarding why some heads harm the models' predictive performance after executing the finetuning procedure.

Future works could apply and evaluate the proposed method in a broader range of tasks and models, including other than Transformers. Moreover, it would be interesting to compare the models suggested in this dissertation to small models like ALBERT (LAN et al., 2020)(an ALBERT configuration similar to BERT-large has 18x fewer parameters and can be trained about 1.7x faster and still achieve significant improvements in several representative downstream tasks) and to models produced using different pruning strategies. Another promising future path to model pruning and understanding is to investigate why some heads do not contribute to or even harm the model predictive performance when we execute the finetuning procedure, mainly to small datasets. Also, it is interesting to evaluate if the size of input sentences influences the number of parameters that Transformer-based models must have to tackle NLP tasks successfully.

Ethic Statement All the datasets contemplated in this dissertation were compiled from previous work that made them publicly available. Even though we have not collected any tweets by ourselves, we comprehend that using data from the Twitter platform should evoke ethical reflections. Twitter users presume their posts are not private; however, they are typically not explicitly informed that their tweets can be used for scientific purposes. Moreover, they usually consider that every tweet is transitory whereas it can be collected and stored by anyone anywhere. We did our best not to include sensitive content in our examples and not disclose their authors' identities.

REFERENCES

BAAN, Joris; HOEVE, Maartje ter; WEES, Marlies van der; SCHUTH, Anne; RIJKE, Maarten de. Understanding Multi-Head Attention in Abstractive Summarization. *CoRR*, abs/1911.03898, 2019. arXiv: 1911.03898. Available from: <http://arxiv.org/abs/1911.03898>.

BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. In: 3RD International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. [S.l.: s.n.], 2015.

BARRETO, Sérgio; MOURA, Ricardo; CARVALHO, Jonnathan; PAES, Aline; PLASTINO, Alexandre. Sentiment analysis in tweets: an assessment study from classical to modern word representation models. *Data Mining and Knowledge Discovery*, Springer, p. 1–63, 2022.

BENDER, Emily M.; GEBRU, Timnit; MCMILLAN-MAJOR, Angelina; SHMITCHELL, Shmargaret. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In: PROCEEDINGS of the 2021 ACM Conference on Fairness, Accountability, and Transparency. Virtual Event, Canada: Association for Computing Machinery, 2021. (FAccT '21), p. 610–623. ISBN 9781450383097.

BROWN, Tom B.; MANN, Benjamin; RYDER, Nick; SUBBIAH, Melanie; KAPLAN, Jared; DHARIWAL, Prafulla; NEELAKANTAN, Arvind; SHYAM, Pranav; SASTRY, Girish; ASKELL, Amanda; AGARWAL, Sandhini; HERBERT-VOSS, Ariel; KRUEGER, Gretchen; HENIGHAN, Tom; CHILD, Rewon; RAMESH, Aditya; ZIEGLER, Daniel M.; WU, Jeffrey; WINTER, Clemens; HESSE, Christopher; CHEN, Mark; SIGLER, Eric; LITWIN, Mateusz; GRAY, Scott; CHESS, Benjamin; CLARK, Jack; BERNER, Christopher; MCCANDLISH, Sam; RADFORD, Alec; SUTSKEVER, Ilya; AMODEI, Dario. Language Models are Few-Shot Learners. In: ADVANCES in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual. [S.l.: s.n.], 2020. Available from: <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html>.

CARVALHO, Jonnathan; PLASTINO, Alexandre. On the evaluation and combination of state-of-the-art features in Twitter sentiment analysis. *Artif. Intell. Rev.*, v. 54, n. 3, p. 1887–1936, 2021.

CHEN, Tianlong; FRANKLE, Jonathan; CHANG, Shiyu; LIU, Sijia; ZHANG, Yang; WANG, Zhangyang; CARBIN, Michael. The Lottery Ticket Hypothesis for Pre-trained BERT Networks. In: ADVANCES in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual. [S.l.: s.n.], 2020. Available from: <https://proceedings.neurips.cc/paper/2020/hash/b6af2c9703f203a2794be03d443af2e3-Abstract.html>.

CLARK, Kevin; KHANDELWAL, Urvashi; LEVY, Omer; MANNING, Christopher D. What Does BERT Look at? An Analysis of BERT's Attention. In: LINZEN, Tal; CHRUPALA, Grzegorz; BELINKOV, Yonatan; HUPKES, Dieuwke (Eds.). *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@ACL 2019, Florence, Italy, August 1, 2019*. [S.l.]: Association for Computational Linguistics, 2019. P. 276–286. DOI: 10.18653/v1/W19-4828. Available from: <https://doi.org/10.18653/v1/W19-4828>.

CORTES, Corinna; VAPNIK, Vladimir. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995.

DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. 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, June 2019. P. 4171–4186. DOI: 10.18653/v1/N19-1423.

DING, Yanzhuo; LIU, Yang; LUAN, Huanbo; SUN, Maosong. Visualizing and Understanding Neural Machine Translation. In: PROCEEDINGS of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vancouver, Canada: Association for Computational Linguistics, July 2017. P. 1150–1159. DOI: 10.18653/v1/P17-1106. Available from: <https://aclanthology.org/P17-1106>.

DODGE, Jesse; ILHARCO, Gabriel; SCHWARTZ, Roy; FARHADI, Ali; HAJISHIRZI, Hannaneh; SMITH, Noah A. Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping. *CoRR*, abs/2002.06305, 2020. arXiv: 2002.06305. Available from: <https://arxiv.org/abs/2002.06305>.

FAN, Angela; GRAVE, Edouard; JOULIN, Armand. Reducing Transformer Depth on Demand with Structured Dropout. In: 8TH International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. [S.l.]: OpenReview.net, 2020. Available from: <https://openreview.net/forum?id=Syl02yStDr>.

FRANKLE, Jonathan; CARBIN, Michael. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In: 7TH International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. [S.l.]: OpenReview.net, 2019. Available from: <https://openreview.net/forum?id=rJl-b3RcF7>.

GORDON, Mitchell A.; DUH, Kevin; ANDREWS, Nicholas. Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning. In: PROCEEDINGS of the 5th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2020, Online, July 9, 2020. [S.l.]: Association for Computational Linguistics, 2020. P. 143–155. DOI: 10.18653/v1/2020.repl4nlp-1.18. Available from: <https://doi.org/10.18653/v1/2020.repl4nlp-1.18>.

GURURANGAN, Suchin; MARASOVIĆ, Ana; SWAYAMDIPTA, Swabha; LO, Kyle; BELTAGY, Iz; DOWNEY, Doug; SMITH, Noah A. Don't Stop Pretraining: Adapt Language Models to Domains and Tasks. In: PROCEEDINGS of the 58th Annual Meeting of the Association for Computational Linguistics. Online: Association for Computational Linguistics, July 2020. P. 8342–8360. DOI: 10.18653/v1/2020.acl-main.740. Available from: <https://aclanthology.org/2020.acl-main.740>.

HAN, Wenjuan; PANG, Bo; WU, Ying Nian. Robust Transfer Learning with Pretrained Language Models through Adapters. In: PROCEEDINGS of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers). Online: Association for Computational Linguistics, Aug. 2021. P. 854–861. DOI: 10.18653/v1/2021.acl-short.108. Available from: <https://aclanthology.org/2021.acl-short.108>.

HINTON, Geoffrey E.; VINYALS, Oriol; DEAN, Jeffrey. Distilling the Knowledge in a Neural Network. *CoRR*, abs/1503.02531, 2015. arXiv: 1503.02531. Available from: <http://arxiv.org/abs/1503.02531>.

HOWARD, Jeremy; RUDER, Sebastian. Universal Language Model Fine-tuning for Text Classification. In: PROCEEDINGS of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Melbourne, Australia: Associa-

tion for Computational Linguistics, July 2018. P. 328–339. DOI: 10.18653/v1/P18-1031. Available from: <https://aclanthology.org/P18-1031>.

JAWAHAR, Ganesh; SAGOT, Benoît; SEDDAH, Djamé. What Does BERT Learn about the Structure of Language? In: PROCEEDINGS of the 57th Annual Meeting of the Association for Computational Linguistics. Florence, Italy: Association for Computational Linguistics, July 2019. P. 3651–3657. DOI: 10.18653/v1/P19-1356. Available from: <https://aclanthology.org/P19-1356>.

K, Karthikeyan; WANG, Zihan; MAYHEW, Stephen; ROTH, Dan. Cross-Lingual Ability of Multilingual BERT: An Empirical Study. In: 8TH International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. [S.l.]: OpenReview.net, 2020. Available from: <https://openreview.net/forum?id=HJeT3yrtDr>.

KAO, Wei-Tsung; WU, Tsung-Han; CHI, Po-Han; HSIEH, Chun-Cheng; LEE, Hung-Yi. *BERT's output layer recognizes all hidden layers? Some Intriguing Phenomena and a simple way to boost BERT*. [S.l.]: arXiv, 2020. DOI: 10.48550/ARXIV.2001.09309. Available from: <https://arxiv.org/abs/2001.09309>.

KOVALEVA, Olga; ROMANOV, Alexey; ROGERS, Anna; RUMSHISKY, Anna. Revealing the Dark Secrets of BERT. In: PROCEEDINGS of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Hong Kong, China: Association for Computational Linguistics, Nov. 2019. P. 4365–4374. DOI: 10.18653/v1/D19-1445. Available from: <https://aclanthology.org/D19-1445>.

LAN, Zhenzhong; CHEN, Mingda; GOODMAN, Sebastian; GIMPEL, Kevin; SHARMA, Piyush; SORICUT, Radu. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In: 8TH International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. [S.l.]: OpenReview.net, 2020. Available from: <https://openreview.net/forum?id=H1eA7AEtvS>.

LIU, Bing. *Sentiment analysis: Mining opinions, sentiments, and emoticons*. [S.l.]: Cambridge University Press, 2020.

LIU, Yijia; CHE, Wanxiang; WANG, Yuxuan; ZHENG, Bo; QIN, Bing; LIU, Ting. Deep Contextualized Word Embeddings for Universal Dependency Parsing. *ACM Trans. Asian Low Resour. Lang. Inf. Process.*, v. 19, n. 1, 9:1–9:17, 2020. DOI: 10.1145/3326497. Available from: <https://doi.org/10.1145/3326497>.

LIU, Yinhan; OTT, Myle; GOYAL, Naman; DU, Jingfei; JOSHI, Mandar; CHEN, Danqi; LEVY, Omer; LEWIS, Mike; ZETTLEMOYER, Luke; STOYANOV, Veselin. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692, 2019. arXiv: 1907.11692. Available from: <http://arxiv.org/abs/1907.11692>.

LOSHCHILOV, Ilya; HUTTER, Frank. Decoupled Weight Decay Regularization. In: 7TH International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. [S.l.]: OpenReview.net, 2019. Available from: <https://openreview.net/forum?id=Bkg6RiCqY7>.

LOUIZOS, Christos; WELLING, Max; KINGMA, Diederik P. Learning Sparse Neural Networks through L_0 Regularization. In: 6TH International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. [S.l.]: OpenReview.net, 2018. Available from: <https://openreview.net/forum?id=H1Y8hhg0b>.

LUONG, Thang; PHAM, Hieu; MANNING, Christopher D. Effective Approaches to Attention-based Neural Machine Translation. In: PROCEEDINGS of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015. [S.l.]: The Association for Computational Linguistics, 2015. P. 1412–1421.

MICHEL, Paul; LEVY, Omer; NEUBIG, Graham. Are Sixteen Heads Really Better than One? In: ADVANCES in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. [S.l.: s.n.], 2019. P. 14014–14024. Available from: <https://proceedings.neurips.cc/paper/2019/hash/2c601ad9d2ff9bc8b282670cdd54f69f-Abstract.html>.

MIKOLOV, Tomás; SUTSKEVER, Ilya; CHEN, Kai; CORRADO, Gregory S.; DEAN, Jeffrey. Distributed Representations of Words and Phrases and their Compositionality. In: ADVANCES in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States. [S.l.: s.n.], 2013. P. 3111–3119.

NGUYEN, Dat Quoc; VU, Thanh; NGUYEN, Anh Tuan. BERTweet: A pre-trained language model for English Tweets. In: PROCEEDINGS of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. [S.l.: s.n.], 2020. P. 9–14.

PAN, Sinno Jialin; YANG, Qiang. A Survey on Transfer Learning. *IEEE Trans. Knowl. Data Eng.*, v. 22, n. 10, p. 1345–1359, 2010.

PASZKE, Adam; GROSS, Sam; MASSA, Francisco; LERER, Adam; BRADBURY, James; CHANAN, Gregory; KILLEEN, Trevor; LIN, Zeming; GIMELSHEIN, Natalia; ANTIGA, Luca; DESMAISON, Alban; KOPF, Andreas; YANG, Edward; DEVITO, Zachary; RAISON, Martin; TEJANI, Alykhan; CHILAMKURTHY, Sasank; STEINER, Benoit; FANG, Lu; BAI, Junjie; CHINTALA, Soumith. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: ADVANCES in Neural Information Processing Systems 32. [S.l.]: Curran Associates, Inc., 2019. P. 8024–8035. Available from: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

PENNINGTON, Jeffrey; SOCHER, Richard; MANNING, Christopher. GloVe: Global Vectors for Word Representation. In: PROCEEDINGS of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, Oct. 2014. P. 1532–1543. DOI: 10.3115/v1/D14-1162. Available from: <https://aclanthology.org/D14-1162>.

PRASANNA, Sai; ROGERS, Anna; RUMSHISKY, Anna. When BERT Plays the Lottery, All Tickets Are Winning. In: PROCEEDINGS of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020. [S.l.]: Association for Computational Linguistics, 2020. P. 3208–3229. DOI: 10.18653/v1/2020.emnlp-main.259. Available from: <https://doi.org/10.18653/v1/2020.emnlp-main.259>.

PRESS, Ofir; SMITH, Noah A.; LEVY, Omer. Improving Transformer Models by Reordering their Sublayers. In: PROCEEDINGS of the 58th Annual Meeting of the Association for Computational Linguistics. Online: Association for Computational Linguistics, July 2020. P. 2996–3005. DOI: 10.18653/v1/2020.acl-main.270. Available from: <https://aclanthology.org/2020.acl-main.270>.

RADFORD, Alec; WU, Jeffrey; CHILD, Rewon; LUAN, David; AMODEI, Dario; SUTSKEVER, Ilya. Language Models are Unsupervised Multitask Learners, 2018. Available from:

<https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf>.

ROSSET, Corby. *Turing-NLG: A 17-billion-parameter language model by Microsoft*. [S.l.]: Microsoft, Feb. 2020. Available from: <https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/>.

RUDER, Sebastian; PETERS, Matthew E; SWAYAMDIPTA, Swabha; WOLF, Thomas. Transfer Learning in Natural Language Processing. In: PROCEEDINGS of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials. [S.l.: s.n.], 2019. P. 15–18.

SAJJAD, Hassan; DALVI, Fahim; DURRANI, Nadir; NAKOV, Preslav. On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language*, Elsevier BV, v. 77, p. 101429, Jan. 2023. DOI: 10.1016/j.csl.2022.101429. Available from: <https://doi.org/10.1016%5C%2Fj.csl.2022.101429>.

SAJJAD, Hassan; DALVI, Fahim; DURRANI, Nadir; NAKOV, Preslav. Poor Man's BERT: Smaller and Faster Transformer Models. *CoRR*, abs/2004.03844, 2020. arXiv: 2004.03844. Available from: <https://arxiv.org/abs/2004.03844>.

SANH, Victor; WOLF, Thomas; RUSH, Alexander M. Movement Pruning: Adaptive Sparsity by Fine-Tuning. In: ADVANCES in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual. [S.l.: s.n.], 2020. Available from: <https://proceedings.neurips.cc/paper/2020/hash/eae15aabaa768ae4a5993a8a4f4fa6e4-Abstract.html>.

SHEN, Sheng; DONG, Zhen; YE, Jiayu; MA, Linjian; YAO, Zhewei; GHOLAMI, Amir; MAHONEY, Michael W.; KEUTZER, Kurt. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. In: THE Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020. [S.l.]: AAAI Press, 2020. P. 8815–8821. Available from: <https://aaai.org/ojs/index.php/AAAI/article/view/6409>.

STRUBELL, Emma; GANESH, Ananya; MCCALLUM, Andrew. Energy and Policy Considerations for Deep Learning in NLP. In: PROCEEDINGS of the 57th Annual Meeting of the Association for Computational Linguistics. [S.l.]: Association for Computational Linguistics, 2019. P. 3645–3650.

SUN, Chi; QIU, Xipeng; XU, Yige; HUANG, Xuanjing. How to Fine-Tune BERT for Text Classification? In: CHINESE Computational Linguistics - 18th China National Conference, CCL 2019, Kunming, China, October 18-20, 2019, Proceedings. [S.l.]: Springer, 2019. v. 11856. (Lecture Notes in Computer Science), p. 194–206. DOI: 10.1007/978-3-030-32381-3_16. Available from: <https://doi.org/10.1007/978-3-030-32381-3%5C_16>.

TENNEY, Ian; DAS, Dipanjan; PAVLICK, Ellie. BERT Rediscovers the Classical NLP Pipeline. In: PROCEEDINGS of the 57th Annual Meeting of the Association for Computational Linguistics. Florence, Italy: Association for Computational Linguistics, July 2019. P. 4593–4601. DOI: 10.18653/v1/P19-1452. Available from: <https://aclanthology.org/P19-1452>.

TURNEY, P. D.; PANTEL, P. From Frequency to Meaning: Vector Space Models of Semantics. *Journal of Artificial Intelligence Research*, AI Access Foundation, v. 37, p. 141–188, Feb. 2010. ISSN 1076-9757. DOI: 10.1613/jair.2934. Available from: <http://dx.doi.org/10.1613/jair.2934>.

VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. Attention is All you Need. In: ADVANCES in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. [S.l.: s.n.], 2017. P. 5998–6008. Available from: <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>.

VIRTANEN, Pauli; GOMMERS, Ralf; OLIPHANT, Travis E.; HABERLAND, Matt; REDDY, Tyler; COURNAPEAU, David; BUROVSKI, Evgeni; PETERSON, Pearu; WECKESSER, Warren; BRIGHT, Jonathan; VAN DER WALT, Stéfan J.; BRETT, Matthew; WILSON, Joshua; MILLMAN, K. Jarrod; MAYOROV, Nikolay; NELSON, Andrew R. J.; JONES, Eric; KERN, Robert; LARSON, Eric; CAREY, C J; POLAT, İlhan; FENG, Yu; MOORE, Eric W.; VANDERPLAS, Jake; LAXALDE, Denis; PERKTOLD, Josef; CIMRMAN, Robert; HENRIKSEN, Ian; QUINTERO, E. A.; HARRIS, Charles R.; ARCHIBALD, Anne M.; RIBEIRO, Antônio H.; PEDREGOSA, Fabian; VAN MULBREGT, Paul; SCIPY 1.0 CONTRIBUTORS. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, v. 17, p. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

VOITA, Elena; TALBOT, David; MOISEEV, Fedor; SENNRICH, Rico; TITOV, Ivan. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. In: PROCEEDINGS of the 57th Annual Meeting of the Association

for Computational Linguistics. Florence, Italy: Association for Computational Linguistics, July 2019. P. 5797–5808. DOI: 10.18653/v1/P19-1580. Available from: <https://aclanthology.org/P19-1580>.

WILLIAMS, Adina; NANGIA, Nikita; BOWMAN, Samuel. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In: PROCEEDINGS of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers). New Orleans, Louisiana: Association for Computational Linguistics, June 2018. P. 1112–1122. DOI: 10.18653/v1/N18-1101. Available from: <https://aclanthology.org/N18-1101>.

WOLF, Thomas; DEBUT, Lysandre; SANH, Victor; CHAUMOND, Julien; DELANGUE, Clement; MOI, Anthony; CISTAC, Pierric; RAULT, Tim; LOUF, Rémi; FUNTOWICZ, Morgan; DAVISON, Joe; SHLEIFER, Sam; PLATEN, Patrick von; MA, Clara; JERNITE, Yacine; PLU, Julien; XU, Canwen; SCAO, Teven Le; GUGGER, Sylvain; DRAME, Mariama; LHOEST, Quentin; RUSH, Alexander M. Transformers: State-of-the-Art Natural Language Processing. In: PROCEEDINGS of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Online: Association for Computational Linguistics, Oct. 2020. P. 38–45. Available from: <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.

WU, Yonghui; SCHUSTER, Mike; CHEN, Zhifeng; LE, Quoc V.; NOROUZI, Mohammad; MACHEREY, Wolfgang; KRIKUN, Maxim; CAO, Yuan; GAO, Qin; MACHEREY, Klaus; KLINGNER, Jeff; SHAH, Apurva; JOHNSON, Melvin; LIU, Xiaobing; KAISER, Lukasz; GOUWS, Stephan; KATO, Yoshikiyo; KUDO, Taku; KAZAWA, Hideto; STEVENS, Keith; KURIAN, George; PATIL, Nishant; WANG, Wei; YOUNG, Cliff; SMITH, Jason; RIESA, Jason; RUDNICK, Alex; VINYALS, Oriol; CORRADO, Greg; HUGHES, Macduff; DEAN, Jeffrey. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*, abs/1609.08144, 2016. arXiv: 1609.08144. Available from: <http://arxiv.org/abs/1609.08144>.

ZAFRIR, Ofir; BOUDOUKH, Guy; IZSAK, Peter; WASSERBLAT, Moshe. Q8BERT: Quantized 8Bit BERT. In: 2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS). [S.l.: s.n.], 2019. P. 36–39. DOI: 10.1109/EMC2-NIPS53020.2019.00016.

ZHU, Y.; KIROS, R.; ZEMEL, R.; SALAKHUTDINOV, R.; URTASUN, R.; TORRALBA, A.; FIDLER, S. Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books. In: 2015 IEEE International Conference on

Computer Vision (ICCV). Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2015. P. 19–27. DOI: 10.1109/ICCV.2015.11. Available from: <https://doi.ieeecomputersociety.org/10.1109/ICCV.2015.11>.