Explainable attention pruning: a meta-learning-based approach

Praboda Rajapaksha, Noel Crespi

Explainable Attention Pruning: A Meta-learning-based Approach

Praboda Rajapaksha, *Member, IEEE,* Noel Crespi, *Senior Member, IEEE*

***Abstract*—Pruning, as a technique to reduce the complexity and size of Transformer-based models, has gained significant attention in recent years. While various models have been successfully pruned, pruning BERT poses unique challenges due to their fine-grained structure and overparameterization. However, by carefully considering these factors, it is possible to prune BERT without significantly degrading its pre-trained loss. In this paper, we propose a Meta-learning-based pruning approach that can adaptively identify and eliminate insignificant attention weights. The performance of the proposed model is compared with several baseline models, as well as the default fine-tuned BERT model. The baseline pruning strategies employed low-level pruning techniques, targeting the removal of only 20% of the connections. The experimental results show that the proposed model outperforms the other baseline models, in terms of lower inference latency, higher MCC and lower loss. However, there is no significant improvement observed in terms of average FLOPs (floating-point operations per second). Furthermore, we conduct a comparative evaluation of the baseline models and our proposed model using two explainable (XAI) approaches. While other models allocate reasonable attention to less significant words for sentiment classification, our model assigns higher probabilities to the most significant sentimental words.**

***Impact Statement*—Efficient handling of inference time in pre-trained language models (PLMs) and the preservation of performance while reducing their size are important research considerations. Model compression techniques, such as pruning, are recognized as effective approaches for achieving memory-efficient, energy-efficient, computation-efficient, and storage-efficient PLMs. Pruning addresses the need to create compact models without compromising their overall effectiveness. Existing pruning methods often rely on task and domain-specific approaches and therefore, it is important to explore a domain-independent pruning approach. We propose a new pruning strategy called Meta-Controller-based Attention Pruning (MCAP) for the BERT model targeting single-sentence prediction tasks. MCAP optimization strategy eliminates insignificant attention in the BERT by calculating their importance scores. The self-supervised pruner in MCAP uses a meta-learning approach to identify and eliminate these insignificant attentions before fine-tuning. Our study compares MCAP with baseline models (both structured and unstructured pruning) and compared it with inference latency, MCC, and loss parameters. The results show that MCAP outperforms the baseline models in terms of inference latency, MCC, and loss. Explainable AI (XAI) techniques are used to interpret the model's decisions and predictions. MCAP focuses on significant words in sentiment classification, ensuring important model parameters are retained without a significant impact on output.**

***Index Terms*—attention, BERT, meta-learning, model compression, pruning, sentiment analysis, transfer learning, XAI**

Praboda Rajapaksha and Noel Crespi are with the Samovar, Telecom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France {praboda.rajapaksha, noel.crespi}@telecom-sudparis.eu

I. Introduction

Over the last four years, pretrained language models such as BERT and RoBERTa improved the performances of various Natural Language Processing (NLP) tasks. Following this, large language models become popular in many research activities and their sheer size limits deploying them in production experiencing higher inference time. Important factors in achieving efficient inference include compact memory footprint (memory-efficient), less disk space (storage-efficient), fewer computational measures (computation-efficient), low inference latency (time-efficient) and less energy consumption (energy-efficient) [5]. One solution to address these factors is to reduce the model size without affecting its performance.

There have been many works on compressing a large model into a lightweight model [1] [5]. Quantization is one approach that can be used during training and inference, and many previous quantization techniques focus on reducing inference time while maintaining significant accuracy [2]. Knowledge distillation is another technique which transfers knowledge from a larger model (teacher) to a smaller one (student) [3]. Pruning is another approach to reduce the pre-trained model size by removing minimally affected neurons, weights or any other parameter in the pre-trained model. Unlike other compression approaches, pruning removes unnecessary connections in the network, and the reduction of unnecessary over-parameters helps to achieve memory-efficient, energy-efficient, computation-efficient and also storage-efficient models. Hence, we propose a novel pruning strategy to achieve efficient inference.

The identification of optimal parameters to prune is challenging, especially when dealing with over-parameterized transformer-based language models [7] [8]. Any Pre-trained language model (PLM) follows three steps: pre-training, fine-tuning and inference and they can be pruned during fine-tuning [9] or after fine-tuning [6]. The problem with these approaches is that the model becomes progressively sparser and weight values are predetermined by the end task. Few attempts are available to compress a model before fine-tuning. A major advantage over pruning before fine-tuning is that it helps to reduce the computational overhead during both fine-tuning and inference and can be generalized to multiple tasks that use similar contextual representations. Previous studies have shown that fine-tuning BERT on a specific task does not enhance its ability to be pruned effectively [26]. In addition, many pruning approaches are Task and domain-dependent and therefore, implementing a domain-independent pruning approach is also important to generalize the optimization

procedure to any downstream task. By initializing the pruning approach before fine-tuning, it becomes possible to generate a domain-independent pruned model.

In this study, we propose the Meta-Controller-based Attention Pruning (MCAP) method that aims to prune the BERT model specifically for single sentence prediction tasks, where it can also adapt to sentence generation tasks as well. MCAP employs a pruning optimization strategy to eliminate insignificant attentions in the BERT transformer by calculating their importance scores and removing those deemed insignificant. The self-supervised pruner in MCAP adopts a meta-learning approach to effectively identify and eliminate these insignificant attentions before fine-tuning. We compare MCAP with several baseline models that utilize both structured and unstructured pruning techniques. To evaluate the impact of our proposed meta-learning paradigm on preserving text representation, we utilize explainable AI (XAI) techniques to interpret the model's decisions and predictions. This allows us to gain valuable insights into the inner workings of the model, ensuring transparency in its decision-making process. The experimental results demonstrate that MCAP exhibits lower inference latency, higher MCC (Matthews Correlation Coefficient), and lower loss in comparison to the other baseline models. Nevertheless, the average FLOPs (Floating Point Operations) improvement of MCAP has not shown any significant advancements compared to the baseline models. Furthermore, when conducting sentiment classification through XAI approaches (LIME and SHAP) on each model, MCAP assigns higher probabilities to the most significant words that play a crucial role in determining the overall sentiment of the sentence, whereas other models give considerable attention to the less sentimental words in the sentence. This approach guarantees to retain important model parameters without affecting much on its output.

II. Background and Methodology

In this section, we provide various cutting-edge approaches for pruning language models, with a specific focus on BERT. We then present our novel pruning optimization strategy, which is based on a meta-controller-based approach.

A. Pruning Language Models

Pruning can be performed either in a structured or unstructured way and, in each scenario, it is required to define some criteria to remove connections by minimally affecting the model performances. In unstructured pruning, optimization approaches find and remove less salient connections in the model, mainly by setting their weight connections to 0. The main objective of unstructured pruning is to remove low-magnitude weights from the weight matrix by selecting unimportant weights through various criteria. The main drawback of this approach is that it produces sparse matrices after pruning [28]. In contrast, structured pruning prunes away a large part of the network such as neurons, attention heads, layers and weight matrix blocks [4]. Hence, unstructured pruning does not pay attention to any relationship between pruning parameters while structured pruning focuses on pruning parameters in groups. Structured pruning [29] requires understanding the model parameters to remove some portion of unit [7], [8]. Some research works focused on direct layer droppings from the language models [27]. Language models use different structured and unstructured pruning criteria such as weight pruning [15], [18], movement pruning [10], block pruning [28], heads pruning and layer pruning [27]. With these approaches, pruning language models have yielded promising results and State-of-the-art techniques have demonstrated significant advancements and improvements.

B. Identifying important parameters for pruning

Deep model pipeline [12] involves a series of well-defined steps in the creation, deployment, monitoring and improvement of a model. Each step in the pipeline makes a specific model output which is independent and can be optimized through certain strategies. The final three stages, collectively known as model inference, is a challenging task as it involves utilizing real-time, unseen data to generate an output in a production environment. Model inference uses a trained model to infer output from unseen data and therefore, when the number of connections in the model decreases, then by default, we can observe faster model inference. Therefore, one of the best approaches to speed up the inference process is to identify the least salient connections and remove them from the model. In this research, we use a novel model pruning mechanism to identify the important features in the BERT and to remove unimportant parameters to speed up both inference and pre-training while developing an optimized model. The meta-learning strategy helps to compress the pre-trained Transformer-based language model before fine-tuning by identifying important attention heads.

Unlike, many other pruning mechanisms proposed for standard supervised learning models, language models use weight values that are predetermined by the original model and fine-tuned for different downstream tasks. Hence, identifying the salient connections before fine-tuning is useful as it overcomes computational overhead during fine-tuning, and it helps to build a task-independent pruned model. Our proposed approach uses a meta-learning paradigm to remove unnecessary connections from BERT before fine-tuning and the pruning strategy iterates only once. BERT-base uses 12 layers containing 12 attention heads in each layer and model training helps to project input embedding into different representation subspaces in each layer. Our experiments are conducted on the BERT encoder to evaluate the representations for each layer.

Mathematically, pruning calculates a matrix S of important scores and then select the unimportant percentage (x) of weights or attention or any other parameters by the importance, as stated in Equation 1. The pruning criteria depend on which parameters or blocks are considered in the optimization strategy.

$$Top_x(S)_{ij} = \begin{cases} 1, & \text{if } S_{ij} \text{ in top x\%} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

With the filtering strategies, we can effectively create a sparse network by defining a mask $M \in \{0,1\}^{n*n}$ which

uses during the forward pass with the provided input y_i and weight matrix W to generate attention score a_i for each head.

$$a_i = WMy_i \tag{2}$$

The pruning strategies proposed in the previous works are mainly based on optimizing the weight matrix using different criteria that are initialized randomly through defined pre-trained criteria. However, these models are difficult to accelerate due to irregular sparsity, limited model parameterisation and inference overhead [21]. Hence, our objective is to use a structured pruning strategy to prune unused attentions that reduce structured sparsity and overhead.

C. BERT Self-attention

In general, Transformer architecture consists of three attentions: self-attention in the Encoder, Encoder-decoder attention in the Decoder and self-attention in the Decoder [22]. Since BERT use only the vanilla Transformer attention, it has only the encoder attention and its operations can be represented with three input parameters: Query (Q), Key (K) and Value (V), where all these parameters are similar in their structure and represent each word in the sequence by a vector. The self-attention in the Encoder is fed with the positional encoding and input embedding to produce and encode representation for each word in the input sequence allowing to capture both the positional and the contextual parameters of each word. Q, K and V parameters in the self-attention produce an encoded representation for each word in the input and generate an attention score for each word. Each attention module adds its attention score to each word's representation through the Encoder stack. In this work, we identify unimportant attention heads in BERT before fine-tuning and evaluate model performances after fine-tuning.

The term 'attention head' in BERT refers to one of the multiple self-attention mechanisms that are applied to the input data. As shown in Figure 1, BERT-base consists of 12 attention heads in each layer, with each head having a dimension of 768 (i.e., a total of 768 hidden units) and they are responsible for computing a specific subset of the attention weights between the input tokens which are then concatenated and used to compute the final attention output. Through visualization, patterns of attention can be observed, such as attention heads focusing on irrelevant or noisy parts of the input, or attention heads that exhibit similar behavior to other heads. These observations can guide the selection of attention heads for pruning, improving the model's efficiency without sacrificing its predictive power. Each attention head performs a unique form of attention, and the collective contribution of all heads enables the model to capture various types of dependencies between the input sequence. Based on Q, K and V parameters, the attention weights are calculated and reflect how much attention the model should pay to each token when predicting a given task.

The size of a BERT attention head can be computed by dividing the number of hidden units by the number of attention heads. The number of hidden units refers to the neurons in the hidden layer, while the number of attention heads corresponds

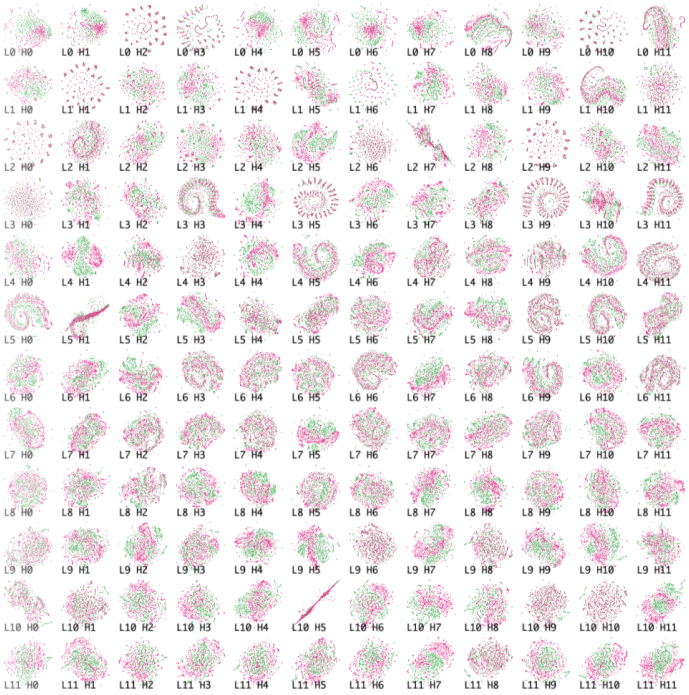

Fig. 1: Joint embedding space for Q-queries (in green) and K-keys (in pink) in Self-attention in BERT. The 12 heads are represented by the columns, while the 12 layers are represented by the rows. The scatterplot depicts each point representing the query or key version of a word, denoted by the point color. Attention head visualization helps in pruning by allowing the identification of redundant or less informative attention heads that can be pruned without significantly affecting the model's performance [30]. Heads with fewer clusters of search results tend to demonstrate more semantic behavior, whereas heads with dispersed results tend to focus more on token position. Hence, pruning will mainly affect the heads that are not capturing semantic and contextual representations as well as heads that are not in the higher layers.

to the attention heads present in the model. For example, in the case of BERT-Base, which has 768 hidden units and 12 attention heads, each attention head's size is 64. This metric gives the dimension of the attention weight vector which is used to calculate the attention score, and this attention score is used to determine the importance of different parts of the input sequence in generating the output. The attention head size is the dimension of the attention weight vector that is used to calculate the attention score and the attention score is used to determine the importance of different parts of the input sequence in generating the output.

Given a sequence of n tokens, $X \in R^{n*d}$ with each token represented by a d-dimensional feature vector, self-attention aims to discover the correlations of all token pairs, where X is first linearly projected into three d_e dimensional spaces and generate Q, K and V as represented in Equation 3.

$$\begin{aligned} Q &= XW_q \in R^{n \times d_e} \\ K &= XW_k \in R^{n \times d_e} \\ V &= XW_v \in R^{n \times d_e} \end{aligned} \tag{3}$$

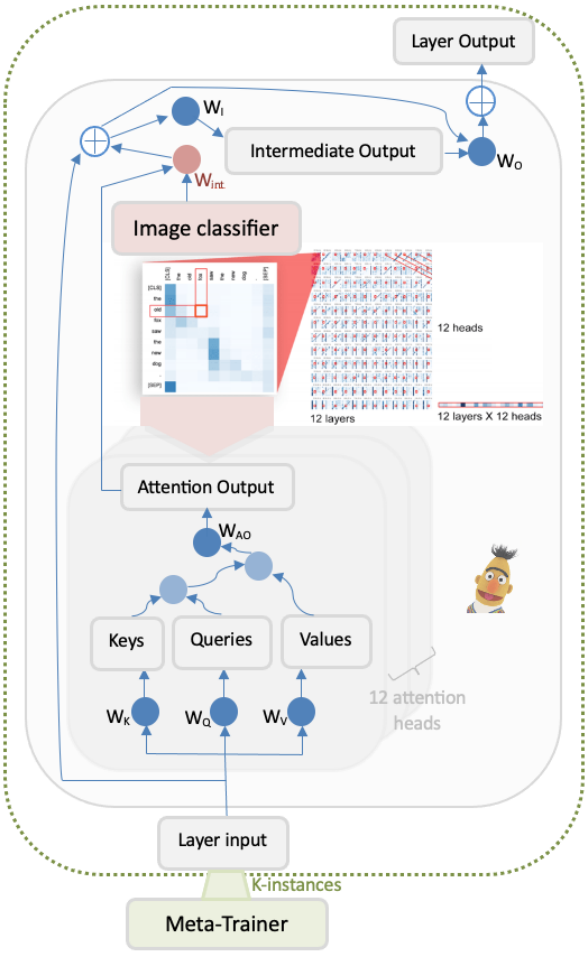

Fig. 2: The MCAP Pruning architecture

where W_q, W_k and $W_v \in R^{n \times d_e}$ are learnable weight matrices, and Q, K and V matrices are of size (batch size, sequence length, d_e).

Self-attention can be expressed in a generic equation as in Equation 4, where $i, j \in 1, .., n$ index the tokens. The self-attention function $\alpha : R^{d_e} \times R^{d_e} \rightarrow R$ is composed of a nonlinear function $\beta : R \rightarrow R$ and a relation function $\gamma : R^{d_e} \times R^{d_e} \rightarrow R$ as in Equation 5.

$$y_{i,:} = \sum_{j=1}^{n} \alpha(Q_{i,:}, K_{j,:}) \cdot V_{j,:} \tag{4}$$

$$\beta(\cdot) = softmax(\cdot)$$
$$\gamma(Q_{i,:}, K_{j,:}) = \frac{1}{\sqrt{(d_e)}} \cdot Q_{i,:}^T K_{j,:}. \tag{5}$$

Therefore, as indicated in Equation 4, the attention weights generated from $\alpha(Q \cdot W)$ are then used to weigh V in the self-attention layer. Hence, the final output of the attention head is a representation of the input sequence that has been weighted by the attention scores. This representation captures the most important relationships between the tokens in the input sequence and thus, it helps the model to attend to different aspects of input for different tasks.

D. Self-attention Map

Self-attention maps are graphical representations of attention weights. They can be used to understand how the model attends to different parts of the input sequence for a given task. There are five frequently occurring patterns of self-attention maps: vertical, diagonal, vertical+diagonal, block and heterogeneous [31]. Except for vertical attention maps, all other attention maps help to interpret linguistic information. These attention maps have been employed in image classification tasks to exploit spatial features within CNNs to identify significant regions within an image and amplify their influence [14]. Hence, we consider this as an image classification task as shown in Figure 2. Therefore, in the context of single-task prediction, CNNs can be leveraged to extract attention pruning scores using attention maps, in which trainable attention modules can be employed, taking 2D feature vector maps (intermediate representations of input images) as inputs and generating relevance scores for each map. These attention maps are integrated with standard CNN architecture with 5 convolution layers. The CNN model is trained to learn the spatial features in the attention maps using 1000 images (400 from [31] and 600 from [14]). This approach has proven effective in previous research, as the learned attention maps effectively highlight regions of interest while disregarding background noise in input images [31]. Inspired by this methodology, we adopt a similar technique to examine the attention map to gain insights into how BERT attends to different parts of the input and which tokens have stronger connections or dependencies. The CNN takes these attention maps as inputs and conducts the feature importance analysis to identify important features in the attention maps and extract attention pruning scores. The goal is to identify salient attention scores to identify the most important attention heads and prune the least important ones with hyperparameter optimization. As shown in Equation 6, the importance score δ_i is generated by passing the output of the CNN to a Sigmoid activation function and concatenates with the matrix encoder to produce a value between 0 and 1. This layer should have the same number of outputs as the number of attention heads in the BERT layers.

$$\delta_i = (\frac{1}{1+e^{-x}})CNN(y_{i,:}) \tag{6}$$

As shown in Figure 2, δ_i value is calculated for each attention i to identify insignificant attentions and then generate a new attention matrix $\hat{y}$ which then will fine-tune for a particular task focusing on the desired loss function. Once we have the importance scores δ_i, as indicated in Figure 2, we generate a new attention matrix by multiplying the original attention matrix by the importance scores. This process gives more importance to the attention heads with higher importance scores.

E. Meta-learning based objective function

As explained, the CNN learns to identify important features in the attention maps and extract attention pruning scores and then, we use a meta-learning process to dynamically adjust

attention weights across different tasks. Meta-learning aims to train our attention pruning model to learn quickly and efficiently on new tasks and the meta-controller is trained on a set of synthetic mini datasets, to learn how to generate attention masks that are task-specific. For each mini dataset, the meta-controller is trained to produce attention masks that maximize the performance of the pruned BERT model on that task. Once the meta-controller is trained, it can be used to generate attention masks for any new task, without the need for further training. To do this, the meta-controller is simply given the new task as input, and it produces a set of attention masks that are tailored to that task.

Let X, Y and T be the input to the model, the output of the model, and the task, respectively. Meta-controller takes the input X from task T and generates attention masks M that will lead to generating the new attention weights of the model. We implemented the meta-controller as a Feed-forward Neural Network, represented in Equation 7, with parameters Φ and θ.

$$\begin{aligned} Y &= f(X, \theta, M) \\ M &= g(T, \Phi) \end{aligned} \tag{7}$$

The function $g(T, \Phi)$ represents the meta-controller's ability to generate attention masks based on the input task T and its parameters Φ, where Φ represents the hyper-parameters of the BERT model. The function $f(X, \theta, M)$ produce an output Y, which is the pruning mask for the BERT model, based on the input X, its parameters θ that represent the parameters of the feed-forward layer in the meta-controller, and the attention masks M. Train the meta-controller to generate pruning masks that minimize the loss of the pruned model on the training tasks by updating the meta-controller's parameters Φ to improve its ability to generate attention masks that improve the performance of the model on the input task T.

In this paper, we mainly focus on single-sentence prediction tasks, but the same procedure can be applied to language generation tasks as well. The meta-controller is trained using two single-sentence prediction datasets in GLUE, namely, SST-2 and CoLA. We split the original training dataset into a 9:1 ratio to generate train and validation datasets. The meta-controller, through meta-learning, learns to produce attention masks that are task-specific. These masks are then used to modify the attention weights obtained from the feature importance analysis. The process begins with pruning applied to subsets (k-instances) of the training instances and then continues iteratively until all instances have been considered. The number of training instances per episode in the meta-controller is set to k=60. In our experiments, we observed the performance of the meta-controller for k=10,20, 30, 60 and 120. Based on the loss and the accuracy of the model, we set the k=60 in our final experiments. The model contains a feed-forward layer with a SGD optimizer and it updates every 8 episodes. Once the importance score for all attentions in each batch has been calculated, we use the Gumbel-softmax to generate a vector representation, which ensures that the modification in attention weights maintains the similarity between the original and pruned models. By taking the task as input, the meta-controller produces a set of new attention masks M that modify the attention weights of the model identifying important scores for each attention to prune the model. It is important to keep the representation of each instance without modifying significantly where the representation before and after pruning should be closer to each other. The representation similarity before and after pruning is based on the internal representation learned by the model. The Equation 8 and 9 represent how to identify the similarity of instances x1, x2, and x3 before and after pruning a BERT model.

$$\begin{aligned} Similarity(x1, x2) &= f(x1, x2, W) \\ Similarity(x1, x3) &= f(x1, x3, W) \end{aligned} \tag{8}$$

where, $x1$, $x2$, and $x3$ represent the instances being compared, f is the function that calculates the representation similarity, W represents the weights of the model, which include the attention weights. The scale of these instances is related to the length of the input sequences, measured in terms of the number of tokens and therefore length of $x1$, $x2$, and $x3$ are 128 tokens. After pruning the BERT model, the equation for the representation similarity of instances $x1$, $x2$, and $x3$ can be represented as follows, where W' represents the pruned weights of the model.

$$\begin{aligned} Similarity(x1, x2) &= f(x1, x2, W') \\ Similarity(x1, x3) &= f(x1, x3, W') \end{aligned} \tag{9}$$

Equation 8 and 9 represents how the representation similarity of instances $x1$, $x2$, and $x3$ is calculated using the same function before and after pruning the BERT model, but the weights used in the calculation are different.

Therefore, our objective function focuses on the relative representation distribution of instances or the normalized distance between one instance with the other instances. The relative distribution of instances in a BERT model using cosine similarity can be represented by Equation 10.

$$CS(x1, x2) = \frac{x1.x2}{||x1|| * ||x2||} \tag{10}$$

where, CS represents the Cosine Similarity, x1 and x2 represent the instances being compared, $x1.x2$ represents the dot product of the instances, $||x1||$ and $||x2||$ represent the L2-norm of the instances. The cosine similarity captures the similarity between x1 and x2, instances by evaluating the cosine of the angle between them in an N-dimensional space, where N corresponds to the number of dimensions in the BERT model by comparing the cosine similarity of the instances' representations learned by the model. In a nutshell, it provides a measure of the similarity between the two instances. Cosine similarity returns a value between -1 and 1, where 1 indicates if both instances are identical, 0 indicates if both instances are orthogonal and -1 indicates if both instances are diametrically opposite. The model examines how the cosine similarity values correspond to the distances between instances and when the cosine similarity between two instances is high, it indicates that they are similar or closely related, while a low cosine similarity suggests dissimilarity.

Consider instance X_n represented as N-dimensional normalized vector r^n, whose i^{th} entry is the relative distance

between X_n and X_i:

$$r_i^n = \frac{Distance(h_n, h_i)}{\Sigma_{j=1}^{N} Distance(h_n, h_j)} \tag{11}$$

The relative distance r_i^n distribution tends to have a smaller distance with the presence of the instances with high cosine similarity, which suggests that the instances in the dataset are well-clustered, and similar instances are grouped closely together. Conversely, if the instances with high cosine similarity exhibit larger distances, it indicates a more scattered or diverse distribution of similar instances.

F. Meta-Controller based Attention Head Pruning - MCAP

We consider our meta-controller-based attention head pruning (MCAP) as an optimization approach. The objective of the MCAP algorithm is to minimize the relative distance r_i^n distribution between before and after pruning. A common objective function is to minimize the Kullback-Leibler (KL) divergence [11] between the relative distance distributions associated with the original and pruned model. KL divergence measures the difference between the real distribution and the predicted distribution. If we assume that the vector representation before and after pruning denote as V and $\hat{V}$, then the KL divergence D_{KL} from the true matrix-V to the predictive matrix-$\hat{V}$ can be computed for some instances X as follows.

$$D_{KL}(V \mid \hat{V}) = E_{X \sim \hat{V}} log \left[\frac{V(X)}{\hat{V}(X)} \right] \tag{12}$$

$$D_{KL}(V \mid \hat{V}) = E_{X \sim \hat{V}}[-logV(X)] - H(\hat{V}(X)) \tag{13}$$

$$f_{Objective} = -D_{KL}(p_{original} || p_{pruned}) \tag{14}$$

We can simplify D_{KL} in terms of cross entropy between V and $\hat{V}$ - $E_{X \sim \hat{V}}[-logV(X)]$ and also the entropy of V - $H(\hat{V}(X))$ similar to the above equation. Hence our objective loss function is to minimize the KL Divergence (D_{KL}) to have fewer variations in their relative distribution. The summation over all instances X in the set of instances is considered to calculate the KL divergence between the relative distance distributions associated between the instance and other instances in a set of instances)Batch. KL divergence is a measure of the dissimilarity between two probability distributions, and it is non-negative where a value of 0 means the two distributions are identical. In contrast, a larger value means they are more dissimilar. Based on our objective function $f_{Objective}$, by minimizing the negative KL divergence, we make the relative distance distribution of the pruned model as close as possible to the original model's relative distance distribution considering each batch. Algorithm 1 represents the procedure of using the optimization strategy with MCAP.

Algorithm 1 Self-Supervised Objective Function

Require:

θ - Original attention maps, β - Pruned attention maps, Γ - Cosine Similarity, RDD -Relative Distance Distribution, BERT, Pruned_BERT

procedure MCAP(*cnn_classifier, meta_controller*)
    $\theta \leftarrow$ *BERT.get_attention_maps()*
    $\beta \leftarrow$ *Pruned_BERT.get_attention_maps()*
    $RDD \leftarrow \Gamma(\theta, \beta)$
    $kl\_divergence \leftarrow$ *KL_divergence(RDD)*
    $cnn\_loss \leftarrow$ *cnn_classifier.loss(pruned_BERT)*
    $objective\_loss \leftarrow kl\_divergence + cnn\_loss$
    meta_controller.update_parameters(objective_loss)
    return *objective_loss*
end procedure

III. Experiments

A. Datasets

Since our meta-controller-based pruning model focuses on single-task predictions, we use two datasets from GLUE: the SST-2 (The Stanford Sentiment Treebank) and the CoLA (The Corpus of Linguistic Acceptability) dataset [15]. The SST-2 dataset comprises sentences extracted from movie reviews along with human annotations indicating their sentiment. The objective is to anticipate the sentiment of a given sentence for binary classification. The SST-2 dataset consists of 67K training instances and 1.8K testing instances. The CoLA dataset comprises judgments on the acceptability of English sentences taken from books and articles. Each instance contains a sequence of words labelled, indicating whether it constitutes a grammatically correct English sentence. The standard test set, which includes both in-domain and out-of-domain sections, has been privately labelled. A single performance score is reported for the combined in-domain and out-of-domain sections of the test set. The CoLA dataset consists of 8.5K training instances and 1K testing instances.

B. Pruning strategies

To evaluate our pruning method, we ran six baseline models as explained below including both magnitude pruning and head pruning techniques.

1) Iterative Magnitude Pruning: Iterative Magnitude Pruning is a common approach to start with training a dense network and subsequently removing weights based on a specific criterion, such as magnitude (absolute value) [16]. For optimal results, this process is typically repeated iteratively by alternating between weight pruning and network retraining. In each iteration, the weights with the smallest magnitudes are pruned, which can lead to increased sparsity in the network. The network is then retrained using the remaining weights to recover the lost accuracy due to pruning. This process is repeated for a fixed number of iterations or until a desired level of sparsity is achieved. IMP is an effective method for reducing the size of neural networks and improving their efficiency without significant loss of accuracy. IMP has been used to

TABLE I: Performance comparisons of MCAP pruning model with the baseline models. HeadIMP stands for Head Importance score and Avg FLOPs refer to average Floating Point Operations. The latency indicates the inference latency and loss refers to the cross-entropy loss during the validation.

	SST2				COLA			
Pruning technique	**Latency (ms)**	**FLOPs (Avg)**	**MCC**	**Loss**	**Latency (ms)**	**FLOPs (Avg)**	**MCC**	**Loss**
1. Iterative magnitude pruning	1100	81.41	0.6314	0.4501	631	141.69	0.3623	0.5952
2. L1 unstructured pruning	1520	69.37	0.6204	0.2781	258	345.72	0.3737	0.6608
3. Random magnitude pruning	1520	58.94	0.7524	0.3511	480	137.71	0.2621	0.6797
4. HeadImp: Avg attention to the [SEP] token	373	162.56	0.7145	0.4085	392	154.34	0.2591	0.6327
5. HeadImp: attention entropy	175	346.39	0.6641	0.4948	817	74.24	0.3251	0.6306
6. Random head pruning	191	317.54	0.6367	0.4541	544	112.48	0.3431	0.6168
7. MCAP (Ours)	**124**	243.69	**0.7252**	**0.2103**	**254**	152.08	**0.3869**	**0.5041**
8. BERT fine-tuning	1190	75.01	0.8604	0.2357	1500	59.58	0.5869	0.5123

prune LLMs as well [16], but there are a few limitations such as re-training overhead, dense connections and structural redundancy in the transformer architecture.

2) L1 Unstructured Pruning: L1 unstructured pruning evaluates the magnitude of each parameter in the model based on the absolute values in which the parameters with the lowest magnitudes are considered less influential and removed from the model. This pruning process is usually performed iteratively, gradually increasing the pruning intensity until the desired level of sparsity is achieved. L1 unstructured pruning is based on L1 regularization, which adds a penalty term to the network's loss function that is proportional to the sum of the absolute values of the weights. This encourages the network to learn sparse weight values, with many of them being close to zero. By pruning weights with the smallest magnitude, L1 unstructured pruning can further increase the sparsity of the network and reduce its size, which can lead to faster inference and lower memory requirements. Compared to other pruning methods, L1 unstructured pruning is relatively simple to implement and is effective in reducing the size of neural networks while maintaining high accuracy on various tasks [17]. However, unstructured pruning techniques like L1 pruning do not take into account the inherent structure and dependencies within language models and pruning such a model is computationally expensive.

3) Random Magnitude Pruning: Random Magnitude Pruning is an unstructured pruning technique used for reducing the size of transformer models by randomly removing a certain percentage of weights or connections from the model. Unlike structured pruning, random pruning involves randomly selecting weights or connections to prune without any specific consideration of their magnitude or importance. Random pruning can be used in combination with other compression techniques, such as quantization or knowledge distillation, to further reduce the model size and improve its efficiency. Recent studies have shown that random pruning is effective without significant loss of performance and Liu et al. [18] show that random pruning can achieve comparable compression rates to structured pruning methods while maintaining high accuracy. Due to its computational inefficiency and lack of fine-grained control over the pruning process (i.e., not considering the significance of individual parameters), random pruning is not as effective as Structured pruning and magnitude-based pruning with adaptive thresholds.

4) Heads important score: attention paid to the SEP token: In some studies, head pruning is often considered better than weight pruning for several reasons, including, reduced overfitting, computational efficiency and preservation of the structure of the model [23]. One popular strategy for head pruning is to compute the head importance score relevant to the average attention paid to the SEP token and based on that prune the heads in decreasing order of the attention head importance score. First, it computes the average attention weight of each head to the SEP token over the entire training dataset by extracting the attention weights of each head during training or fine-tuning the model. Next, the attention heads are ranked based on their average attention weight to the SEP token, placing the head that exhibits the highest attention towards the SEP token at the top of the ranking. Then, determine a threshold representing the desired percentage of attention heads to be pruned from the model, considering factors such as the desired level of sparsity or available computational resources. Proceed to remove the attention heads with the lowest average attention weight to the SEP token iteratively until the desired level of sparsity is attained. In the final step, fine-tune the pruned model to recover the lost accuracy due to pruning. However, this pruning strategy has some limitations as it does not necessarily capture the overall importance of the heads in the model because some heads are capable of capturing important linguistic information that is not directly related to the SEP token. Another drawback of pruning heads based on the average attention to the SEP token is that all downstream applications do not rely equally on the SEP token and different tasks can have diverse dependencies on various parts of the input sequence.

5) Heads important score: Attention entropy: Calculating head importance using attention entropy is a common approach that measures the variability of attention weights across different positions in the input sequence. The higher the entropy, the more evenly distributed the attention weights are, indicating a more important and informative head. Pruning based on attention head importance score is a structured pruning technique that computes the importance score for each attention head using an attention entropy by normalising the attention weights for each position and dividing them by the sum of attention weights at that position. Then, calculate the entropy for each position by multiplying the normalized attention weights by their logarithm and summing them up.

Next, the average attention entropy across all positions is calculated for each attention head to measure the head's attention distribution variability. With attention entropy, we can quantitatively assess the importance of attention heads based on the diversity and distribution of their attention weights. Heads with higher entropy values are likely to capture more informative and relevant patterns in the input sequence [24].

6) Random head pruning: Random head pruning completely ignored the notion of calculating and identifying important heads. We can modify random sampling to experiment with different numbers of heads to be pruned and previous studies showed that half of the attention heads of BERT can be randomly pruned without affecting much on the performance [25]. Random pruning can be executed by removing n number of heads in each layer and followed by the fine-tuning of the model with different seeds. In general random head pruning is faster than other pruning strategies since it does not need to conduct any search strategies to select what heads to prune. A major drawback of this pruning strategy is that it can remove important heads that capture specific patterns or dependencies in the input data.

C. Experimental results

The evaluations are performed on Google Colab Pro+. As explained in the previous sections, we use six different pruning strategies and BERT fine-tuning as baseline models to compare our MCAP pruning model. The baseline pruning strategies employed low-level pruning techniques, targeting the removal of only 20% of the connections. We chose low-level pruning as it is the most widely used approach that does not affect much on the pre-training loss [26]. We fine-tune a separate model on SST2 and COLA tasks for three epochs and try four learning rates: [2, 3, 4, 5] × 10-5. The best evaluation accuracies are averaged and the most suitable hyperparameters for each pruned model. We observed that pruning 20% of the attention heads speeds up fine-tuning by more than 22% and reduces the memory overhead per instance by around 18%. We trained the model by pruning 50% of the attention heads, which reduced the overhead, but model classification performances are not as good as 0% pruning, or standard fine-tuning.

To evaluate the performances of each pruning strategy and to compare with MCAP we provide insights about average FLOPs (Floating Point Operations) and latency as quantitative measures to understand runtime characteristics and computational measures. FLOPs compare the computational requirements of different models to estimate the computational cost of running a model on different platforms, representing the number of floating-point operations performed during inference. On the other hand, latency assesses the responsiveness of a model through the time it takes for a model to process a given input and produce the required output during the validation.

As shown in Table I, both SST2 and CoLA it is not arguably perform better in terms of the average FLOPs value. It is important to note that a low FLOP value does not necessarily indicate a better-performing model, as it does not directly correlate with model performance, while high FLOPs indicate a large computational load, which may increase the overall computational cost. However, lower latency is generally desirable as it implies quicker model predictions. In general, there is a positive correlation between latency and FLOPs. This is because more FLOPs typically lead to more complex computations, which can increase the processing time. Our proposed MCAP method achieved low latency while having relatively high FLOPs. This is mainly due to the specific focus of the MCAP being on optimizing the inference latency by reducing the number of computations and parameters required for each layer, while still maintaining sufficient accuracy. This results in a higher FLOP count compared to methods that focus more on accuracy-efficiency trade-offs.

To comprehensively evaluate the model, we considered other parameters such as MCC (Matthews Correlation Coefficient), average loss and also model's latency. These additional metrics provide more meaningful insights into the model's performance. MCC measures the quality of binary classifications, while model loss reflects the discrepancy between predicted and actual values. By considering these factors, we obtained a more comprehensive understanding of the model's overall performance compared to the baseline models. Table I demonstrates that MCAP outperformed other pruning strategies across both the SST2 and CoLA datasets in terms of lower inference latency, higher MCC, and lower average loss compared to other pruning techniques considered in our analysis.

To further investigate the comparative performance of MCAP, we constructed a Figure 3 that visually depicts the relationship between model accuracy and loss. This figure provides a comprehensive understanding of how the model's accuracy varies with the corresponding loss values. For this experiment, we executed each model for 10 epochs. The figure vividly demonstrates that MCAP initiates with a lower loss value even during the first epoch, surpassing the performance of the baseline models. As training progresses, MCAP continues to reduce the loss while simultaneously improving accuracy. Interestingly, by the third epoch, MCAP demonstrates notably higher accuracy compared to all other models in the comparison. Since the validation accuracy continues to improve while the validation error decreases, it indicates that the MCAP-based model is generalizing well to unseen data.

IV. Explainability of the MCAP

To provide more explanation on the MCAP model and to understand the reason behind the efficient predictions, we interpret and explain the model for a set of given textual content. Model explainability provides the shreds of evidence to trust the model, especially in crucial real-life scenarios and enables us to understand models comprehensively helping for debugging and bias mitigation. Hence, we adopted two main Explainable artificial intelligence (XAI) [19] techniques, as explained in the following sections, to understand our model predictions for a given text.

A. Preliminaries

1) Local Interpretable Model—Agnostic Explanations (LIME): LIME [20] stands as a model-agnostic method that

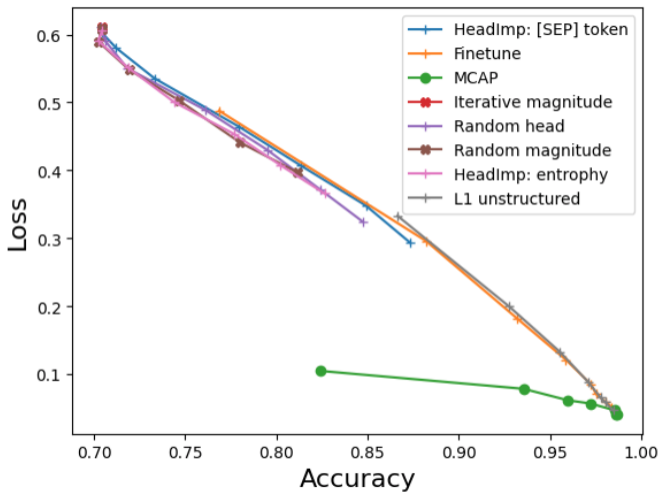

Fig. 3: The relationship between model validation accuracy and validation error. Validation accuracy represents the proportion of correctly predicted instances in the validation dataset, while validation error quantifies the discrepancy between the model's predictions and true labels in the validation dataset.

offers comprehensibility and operates independently of any specific model. It provides an intuitive understanding of the model's behaviour by focusing on local fidelity, ensuring an accurate reflection of its behaviour in the vicinity of a given sample. LIME is one of the first single prediction explainability approaches which selects instances around predictions being explained and modifies them to build a linear model that is inherently interpretable. In the author's original paper [20], the mathematical model has been clearly explained and the following equation provides an abstract view of their model.

Let G be the class of potential explainable models, $g \in G$ which is a model that will be explained to the user $\Omega(g)$ measures the complexity of the model. The locally weighted square loss $\mathcal{L}$ is defined as in the following equation, where $\pi^x(z)$ is an exponential kernel defined on a considered distance function.

$$\mathcal{L}(f, g, \pi_x) = \Sigma_{z,z' \in Z} \pi_x(z)(f(z) - g(z^{'}))^2 \tag{15}$$

The locally weighted square loss $\mathcal{L}(f, g, \pi_x)$ quantifies how g is in approximating the explained model f in the proximity measure between both instances z and sample x. The objective is the minimize $\mathcal{L}(f, g, \pi_x)$ and keep the $\Omega(g)$ in a human-understandable manner. Hence, the explanation ξ with LIME can be interpreted as follows.

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{16}$$

The LIME pseudocode is given Algorithm 2. In the text classification task, the interpretable representation is a bag of words with a limited number of K words, in which K can be denoted with different values based on various instances that can be handled by the user.

Algorithm 2 Sparse Linear Explanations tuning LIME [20]

Require: Classifier f Number of samples N
Require: Instance x, and its interpretable version $x^{'}$
Require: Similarity kernel π_x, Length of explanation K
$Z \leftarrow \{\}\ i \in \{1, 2, 3, ...N\}$
$z_i^{'} \leftarrow sample\_around(x^{'})$
$Z \leftarrow Z \cup \langle z_i^{'}, f(z_i, \pi_x(z_i)) \rangle$

$w \leftarrow KLasso(Z, K)$
return w

The LIME explainer, model-agnostic explainability approach, is capable of explaining any black-box classifier with two or more classes, that take in raw text and generate a probability for each class as an output. To simply understand the workings of LIME, initially, it hides a specific word within a sentence (MASK token) and observes the resulting changes in predictions. The perturbed outputs are subsequently assigned weights based on their similarity to the example being explained. By applying this weighting scheme, LIME provide a streamlined understanding of the relationship between perturbed inputs and corresponding predictions.

2) Shapley Additive exPlanations (SHAP): Shapley values are primarily utilized in the literature in conjunction with game theory methodologies that exhibit desirable characteristics[1]. Shapley values have been used in regression models, decision trees, and correlation analysis in the past. More recently, they have also been employed with Transformer-based models as well. To use game theory concepts with SHAP in our analyses, we assume that the final result is the model's predictions in which input words serve as the players involved. As the players likely made varying contributions to the outcome, their payout should reflect their impact.

SHAP is similar to LIME in terms of perturbation strategies, but it calculates the individual contributions that each feature makes to the model's prediction. However, similar to the power set in mathematics, it should determine the importance of a single feature by considering all possible combinations of features. As a result, SHAP approximates all potential models using the provided dataset, which encompasses the input provided in a specific explanation. By considering all possible combinations, SHAP takes into account the cumulative impact of each feature by considering their marginal contributions.

Consider that in a given classifier f and M features, the shapely value for each feature i is generated independently and can be represented as the weighted average of the relative outcome differences across all the features subsets $S \subseteq M \backslash \{i\}$ as in the following equation.

$$f(S \cup \{i\}) - f(S) \tag{17}$$

Together we can identify $2^{\|M\|}$ possible choices for S, but computing exact Shapley values is a task of exponential complexity. The SHAP framework gives several strategies to approximate efficiently and the model we used in our analysis is DeepSHAP.

[1]https://shorturl.at/awxLR

TABLE II: Textual data used for the intepretable analysis.

	Text	Actual Sentiment	Model Prediction
T1	Loving the first days of summer! <3	Positive	Positive
T2	I hate when people put lol when we are having a serious talk.	Negative	Negative
T3	People are complaining about lack of talent. Takes a lot of talent to ignore objectification and degradation #MissAmerica	Neutral	Negative

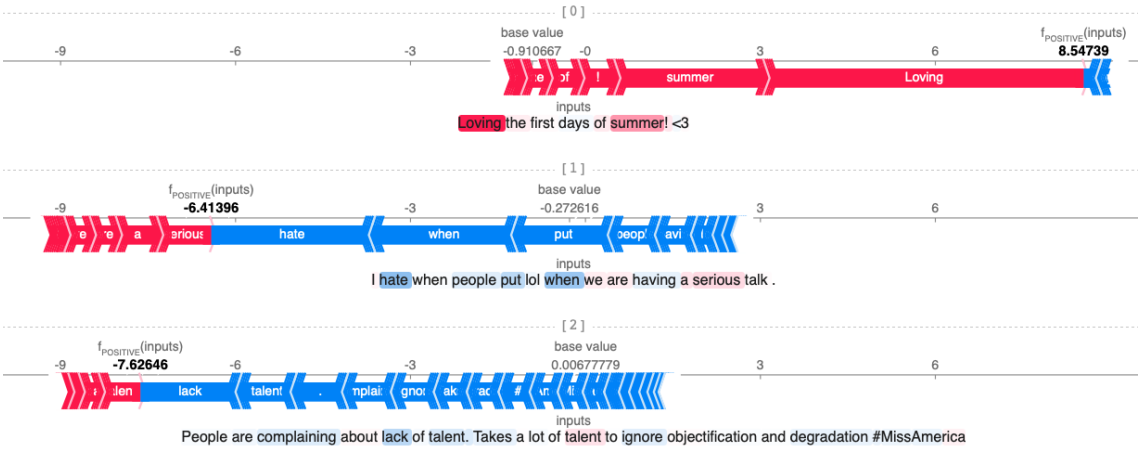

Fig. 4: Visualization of SHAP Representations: Force Plot for Sentiment Analysis Model (Fine-tuned with GLUE SST-2).

B. Explainability with LIME and SHAP

We conduct experiments with LIME and SHAP using our MCAP model to fine-tune with SST2 data. To conduct the analysis, as given in Table II, we selected three distinct texts (T1, T2 and T3) that effectively represent positive, negative, and neutral sentiments. These texts were carefully chosen to provide a comprehensive representation of the various sentiment categories to analyze sentiment classification performance with the BERT model fine-tuned on SST2.

Figure 4 depicts the force plot for SHAP representations for each input T1, T2 and T3. For SHAP analysis, it is necessary to have tensor outputs from the classifier, and explanations are most effective when working in additive spaces. Therefore, we convert the probabilities into logit values (information values) to facilitate the process. Hence, Figure 4 is generated with the obtained logit values for three instances which is specifically crafted to show a comprehensive view of how all the components of the text interact and contribute to the model's output. The base value denotes the average prediction of the model across the entire dataset, while $f(x)$ represents the output probability of the model for the specific instance being analyzed. Red attributes influence the predictions towards class 1 (indicating a positive review), while blue attributes influence the predictions towards class 0 (indicating a negative review). The positive red features exert an upward 'push' on the model's output, while negative blue features exert a downward 'push' on the model's output.

Figure 4 depicts that individual words have a significant impact on model predictions, making word-level contributions a key indicator for detecting sentiments. This is particularly evident in the case of the neutral sentiment text T3, which highlights this aspect even more clearly. The original T1 sample is classified as positive with high confidence, whereas

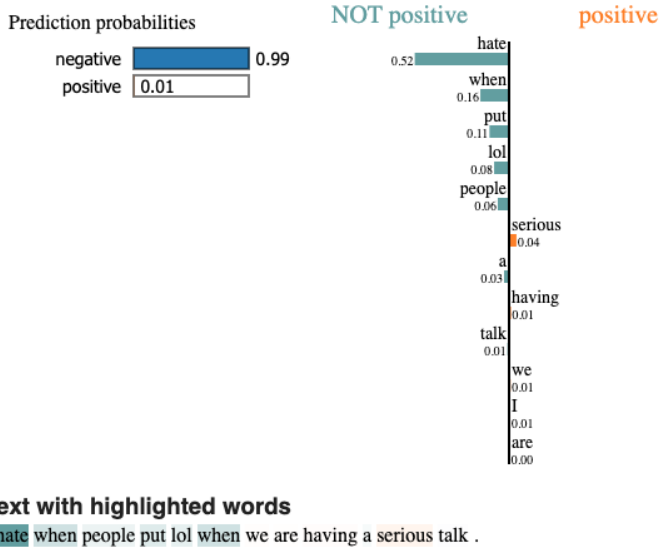

Fig. 5: Visualization of Prediction Explanation using LIME for T2 Model Fine-tuned on GLUE SST-2.

neutral sentiment for T3 is predicted as negative based on the word-level contributions.

Compared with the SHAP model, LIME generates a Lasso tree providing more human-friendly explanations. This enables us to grasp unfamiliar word embedding notations in a more interpretable manner. The explanatory process using LIME for T2 is illustrated in Figure 5. The figure demonstrates that the LIME explainer assigns weights to individual words in the text, indicating their significance in the overall decision-making process. Among the words analyzed, the term *hate* possessed the highest weight (approximately 0.20), signifying

Model	**PP**	$\omega 1$	$\omega 2$	$\omega 4$	$\omega 5$	$\omega 3$	$\omega 11$
BERT Fine-tuning	0.99	(-) 0.52	(-) 0.16	(-) 0.11	(-) 0.08	(-) 0.06	(+) 0.04
Iterative magnitude pruning	0.89	(-) 0.28	(-) 0.26	(-) 0.02	(-) 0.03	(-) 0.01	(+) 0.01
L1 unstructured pruning	0.99	(-) 0.43	(-) 0.25	(-) 0.07	(-) 0.08	(-) 0.05	(+) 0.10
Random magnitude pruning	0.92	(-) 0.30	(-) 0.08	(-) 0.19	(-) 0.10	(+) 0.04	(+) 0.06
HeadImp: Avg attention to [SEP] token	0.96	(-) 0.48	(-) 0.09	(-) 0.02	(-) 0.02	(-) 0.03	(+) 0.08
HeadImp: Attention entropy	0.96	(-) 0.44	(-) 0.09	(-) 0.05	(-) 0.04	(-) 0.03	(+) 0.15
Random head pruning	0.99	(-) 0.38	(-) 0.14	(-) 0.05	(-) 0.10	(-) 0.04	(+) 0.08
MCAP (Ours)	0.97	(-) 0.48	(-) **0.08**	(-) 0.04	(-) 0.07	(+) 0.01	(+) 0.03

TABLE III: The LIME predictions and the most influential word-level contributions to the sentiment prediction probabilities for T2. PP stands for prediction probability. The symbol (-) indicates word-level contribution probabilities for predicting towards negative sentiments, while (+) represents positive contribution probabilities for predicting towards positive sentiments.

its substantial contribution to the overall sentiment of the text. Surprisingly, the presence of the word *when* increased the prediction probability by 0.16 for negative sentiment, despite not providing any additional contextual information related to the negative sentiment. The overall sentiment level of the T2 was 99% and classified as negative.

C. Explainability of pruning methods

We analyzed the explainability of each pruned model to gain a deeper understanding of its inner workings. By examining the explainability of these models, we aimed to shed light on the factors and features that contribute most significantly to their predictions. As shown in the previous section, LIME and SHAP are inherently better than the other for explaining BERT. While SHAP offers global interpretability of the model, it does not yield significant variations in the prediction probabilities. LIME is a model-agnostic, simple and faster algorithm that can provide intuitive explanations for BERT's predictions for textual data. Hence, we use the results generated through LIME for the following comparisons.

In this set of experiments we focus on the results generated for 'T2' and for ease of understanding, we assigned representations from $\omega 0$ to $\omega 12$ for each word in the sentence, '*I hate when people put lol when we are having a serious talk.*'. The prediction probabilities for each model considered in our experiments, including our own, are presented in Table III.

By examining Table III, we observe that the word *hate*-$\omega 1$ makes the most significant contribution to the sentiment classification of the input sentence. However, it is noteworthy that the probabilities associated with this contribution vary across different models. Furthermore, words like *when*-$\omega 2$ are not influential when it comes to sentiment detection. However, it is interesting to note that models like Iterative Pruning and L1 Unstructured Pruning assign a higher level of contribution to such words. In contrast, MCAP assigns less significance to those words and instead prioritizes other words that contribute more effectively to better predictions. We conducted experiments using considerably longer sentences, and the results demonstrate that MCAP focuses on prioritizing the most crucial words for sentiment analysis, resulting in slightly improved predictions compared to other pruning models. When considering the overall LIME probability predictions, it is evident that the fine-tuned model outperformed the others. However, interestingly, the explainability analysis reveals that less influential words receive higher priorities in influencing the predictions.

V. Conclusion and Future Works

The study introduces a model called Meta-Controller-based Attention Pruning (MCAP) that aims to prune the BERT model specifically for single sentence prediction tasks, with potential application to sentence generation tasks. MCAP utilizes a pruning optimization strategy to eliminate insignificant attention heads in the BERT transformer by calculating their importance scores and removing those deemed insignificant. The self-supervised pruner in MCAP adopts a meta-learning approach to effectively identify and eliminate these insignificant heads before fine-tuning. The study compares MCAP with several baseline models that employ structured and unstructured pruning techniques. By employing explainable AI techniques to interpret the model's decisions and predictions, we ensure transparency in the decision-making process. Experimental results demonstrate that MCAP outperforms the baseline models in terms of lower inference latency, higher Matthews Correlation Coefficient (MCC), and lower loss. The primary objective of MCAP is to prune the BERT model for single-sentence prediction tasks while preserving important text representations. The meta-learning paradigm employed by MCAP ensures the removal of insignificant attention, while explainable AI techniques provide insights into the model's decision-making process. Although the average FLOPs improvement of MCAP compared to baseline models is not significant, MCAP assigns higher probabilities to the most significant words in sentiment classification tasks, thereby ensuring the retention of important model parameters without significantly affecting the model's output. Overall, MCAP presents a promising approach for pruning BERT models, improving efficiency, and maintaining performance in single-sentence prediction tasks.

Based on the findings and conclusions of the study, there are several potential future directions for further research and improvement. We aim to generalize the same strategy for language generation tasks as well. Generalizing the approach to a wider range of NLP tasks could provide insights into its effectiveness and potential benefits. While MCAP effectively prunes insignificant attention heads before fine-tuning, exploring different fine-tuning strategies could potentially enhance the performance further. Thereby, we will use a variety of hyperparameters to train our model and compare it with base-

lines by evaluating through large datasets. Efficiency optimization: Although MCAP demonstrates lower inference latency compared to baseline models, further research can be done to optimize its efficiency even more. Investigating techniques like model compression, quantization, or knowledge distillation could potentially reduce the computational resources required by MCAP without sacrificing its performance. By employing attention head visualization techniques, we will conduct further research to identify the most important parameters. Subsequently, our focus will shift towards directly pruning these parameters, eliminating the need for iterative analysis over the entire model.

REFERENCES

[1] Sun, Zhiqing, et al. "MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices." Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020.

[2] Kim, Sehoon, et al. "I-bert: Integer-only bert quantization." International conference on machine learning. PMLR, 2021.

[3] Gou, Jianping, et al. "Knowledge distillation: A survey." International Journal of Computer Vision 129.6 (2021): 1789-1819.

[4] Gupta, Manish, and Puneet Agrawal. "Compression of deep learning models for text: A survey." ACM Transactions on Knowledge Discovery from Data (TKDD) 16.4 (2022): 1-55.

[5] Xu, Canwen, and Julian McAuley. "A survey on model compression for natural language processing." arXiv preprint arXiv:2202.07105 (2022).

[6] Fan, Angela, Edouard Grave, and Armand Joulin. "Reducing transformer depth on demand with structured dropout." arXiv preprint arXiv:1909.11556 (2019).

[7] Michel, Paul, Omer Levy, and Graham Neubig. "Are sixteen heads really better than one?." Advances in neural information processing systems 32 (2019).

[8] Voita, Elena, et al. "Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned." Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. 2019.

[9] Sanh, Victor, Thomas Wolf, and Alexander Rush. "Movement pruning: Adaptive sparsity by fine-tuning." Advances in Neural Information Processing Systems 33 (2020): 20378-20389.

[10] Elesedy, Bryn, Varun Kanade, and Yee Whye Teh. "Lottery tickets in linear models: An analysis of iterative magnitude pruning." arXiv preprint arXiv:2007.08243 (2020).

[11] Kullback, Solomon, and Richard A. Leibler. "On information and sufficiency." The annals of mathematical statistics 22.1 (1951): 79-86.

[12] Mohr, Felix, et al. "Predicting machine learning pipeline runtimes in the context of automated machine learning." IEEE Transactions on Pattern Analysis and Machine Intelligence 43.9 (2021): 3055-3066.

[13] Li, Zhuohan, et al. "Train big, then compress: Rethinking model size for efficient training and inference of transformers." International Conference on machine learning. PMLR, 2020. Mitchell

[14] Prasanna, Sai, Anna Rogers, and Anna Rumshisky. "When BERT Plays the Lottery, All Tickets Are Winning." Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2020.

[15] Wang, Alex, et al. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding." Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP. 2018.

[16] Maene, Jaron, Mingxiao Li, and Marie-Francine Moens. "Towards understanding iterative magnitude pruning: Why lottery tickets win." arXiv preprint arXiv:2106.06955 (2021).

[17] Azarian, Kambiz, et al. "Learned threshold pruning." arXiv preprint arXiv:2003.00075 (2020).

[18] Gale, Trevor, Erich Elsen, and Sara Hooker. "The state of sparsity in deep neural networks." arXiv preprint arXiv:1902.09574 (2019).

[19] Adadi, Amina, and Mohammed Berrada. "Peeking inside the black-box: a survey on explainable artificial intelligence (XAI)." IEEE access 6 (2018): 52138-52160.

[20] Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. "" Why should i trust you?" Explaining the predictions of any classifier." Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. 2016.

[21] Han, S., Pool, J., Tran, J., Dally, W. (2015). Learning both weights and connections for efficient neural network. Advances in neural information processing systems, 28.

[22] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[23] Wang, Hanrui, Zhekai Zhang, and Song Han. "Spatten: Efficient sparse attention architecture with cascade token and head pruning." 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2021.

[24] Clark, Kevin, et al. "What Does BERT Look at? An Analysis of BERT's Attention." Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP. 2019.

[25] Budhraja, Aakriti, et al. "On the weak link between importance and prunability of attention heads." Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2020.

[26] Gordon, Mitchell, Kevin Duh, and Nicholas Andrews. "Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning." Proceedings of the 5th Workshop on Representation Learning for NLP. 2020.

[27] Sajjad, Hassan, et al. "On the effect of dropping layers of pretrained transformer models." Computer Speech Language 77 (2023): 101429.

[28] Lagunas, François, et al. "Block Pruning For Faster Transformers." Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. 2021.

[29] Xia, Mengzhou, Zexuan Zhong, and Danqi Chen. "Structured Pruning Learns Compact and Accurate Models." Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2022.

[30] Yeh, C., Chen, Y., Wu, A., Chen, C., Viégas, F., Wattenberg, M. (2023). AttentionViz: A Global View of Transformer Attention. arXiv preprint arXiv:2305.03210.

[31] Kovaleva, Olga, et al. "Revealing the Dark Secrets of BERT." Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 2019.

Praboda Rajapaksha (Member, IEEE) is a Research Fellow at the Institut Polytechnique de Paris, France. She holds a PhD in Computer Science from the Institut Polytechnique de Paris, France (2021). She received her M.Eng. in Computer Science from the Asian Institute of Technology, Thailand; her M.Sc. in Communication Networks and Services from Institut Mines-Telecom, France and her B.Eng. in Computer Engineering from the University of Peradeniya, Sri Lanka. Her primary research interests include NLP, Deep Learning and Data analysis. She has 7+ years of experience as researcher and collaborated in several European (ITEA, BPI, Horizon) and French national projects (FUI).

Noel Crespi (Senior Member, IEEE) received the master's degree from the Universities of Orsay (Paris 11) and Kent (U.K.), the diplome d'ingénieur degree from the Telecom ParisTech, the Ph.D. and Habilitation degrees from UPMC (Paris-Sorbonne University). Since 1993, he has been working at CLIP, Bouygues Telecom, and then at Orange Labs in 1995. In 1999, he joined Nortel Networks as a Telephony Program Manager, architecting core network products for EMEA region. He joined Institut Mines-Telecom in 2002. He is currently a Professor and the Program Director, leading the Service Architecture Lab. He coordinates the standardization activities for Institut Mines-Telecom at ITU-T and ETSI. He is also an Adjunct Professor at KAIST, South Korea, an Affiliate Professor at Concordia University (Canada), and a Guest Researcher at the University of Goettingen, Germany. He is the Scientific Director of the French-Korean Laboratory ILLUMINE. His current research interests are in data analytics, the IoT, and softwarization.