Less is KEN: a Simple Non-Parametric Pruning Algorithm for Transformers Compression

Anonymous ACL submission

Abstract

Neural network pruning has become increasingly crucial due to the complexity of neural network models and their widespread use in different fields. However, most pruning algorithms are architecture-specific or extremely complex, focusing on elaborate algebraic or geometric calculations for parameter elimination. Additionally, many algorithms are only useful during training or testing phases and reset once the model stops working. This article presents KEN: a simple, unstructured, magnitude-based pruning algorithm based on KDE (Kernel Density Estimator). KEN is designed to create a concentrated transformer model that can be easily injected into its pre-trained version for downstream tasks. We tested KEN on five different transformer models and observed that it performs the same as the original models but with a minimum average weight reduction of 25%. Moreover, we compared KEN with other pruning algorithms and found that it outperforms them, even with fewer parameters. Finally, KEN allows the download of the compact model, reducing memory storage and to inject it into its pre-trained version at any time.

1 Introduction

Large Language Models (LLMs) have become the best and simplest solution for achieving state-of-the-art results in many natural language processing (NLP) applications. However, the increasing use of neural networks (NNs) and transformer models (Vaswani et al., 2017) has resulted in a rise in computational cost due to the complexity of arithmetic calculations, larger matrices, and the addition of more layers. Consequently, the weight of these models and their structures become more complex, leading to a high demand for computation and memory.

One of the best approaches to address the overwhelming size of LLMs is to reduce their resources through *pruning algorithms*. These algorithms can eliminate parameters or entire components in a NN, making it lighter without compromising its original performance. Pruning algorithms emerged in parallel with the earliest use of NNs (Mozer and Smolensky, 1989; Janowsky, 1989; LeCun et al., 1989), but they have gained significant importance in the last decade due to the widespread use of these networks in various fields. There are many pruning algorithms in literature (Blalock et al., 2020), each with a unique approach or adapted old algorithms for these new architectures (Benbaki et al., 2023). However, the complexity of neural networks can pose a challenge when creating pruning algorithms, as these algorithms may require creating new complex theories to make the models lightweight (Dong et al., 2017; Malach et al., 2020). Additionally, most pruning algorithms have several gaps in completeness, as analyzed by Blalock et al., 2020. One important aspect not thoroughly analyzed in almost all pruning algorithms is the storage of the reduced final model. Some of these algorithms only reduce the size of the model during its use, without actually storing it in a lighter form. Therefore, most

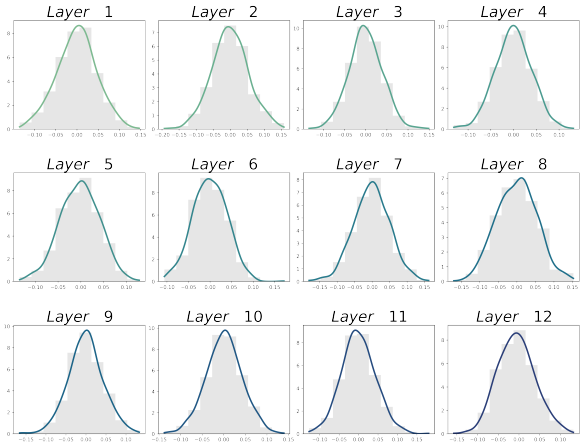


Figure 1: Comparison of the histogram (grey) and its kernel density estimator constructed using the first row of all `pos_q_proj` matrices attention layers on DeBERTa model set to $k = 500$

algorithms in literature focus only on the speed at which they reduce and execute the model without considering this crucial final stage. This is particularly important in resource-limited environments that use neural networks, such as smart devices and mobile phones (Yang et al., 2017; Sze et al., 2017).

In this article, we present KEN (**K**ernel density **E**stimator for **N**eural network compression): a simple but efficient unstructured, magnitude-based KDE pruning algorithm. Using KDE (Kernel density estimator), we can design an abstraction of the analyzed parameters, using solely the *compression value* we want to achieve and the fine-tuned model we want to compress. This implies that KEN produces a *concentrated model* that can be injected into its pre-trained version for downstream tasks. Using KEN, we can define a new pruning approach without trying to minimize the loss function or find the best parameters by examining both pre-trained and fine-tuned versions of a NN. Instead, we focus on generating a lightweight, fine-tuned model that can be easily injected. We evaluated our algorithm by performing it on various tasks on five different transformer models based on a BERT-like architecture (Devlin et al., 2018) and demonstrated that KEN can reduce the size of the models by an average of 25% on a zero-shot regime without affecting the performance of the original NNs. This reduction is also reflected in the weight of the model itself, which decreases proportionally to the compression achieved. Moreover, we compared KEN with other pruning algorithms designed for transformer models, demonstrating the effectiveness of this algorithm. Using KEN, we explained how a non-parametric method widely used in statistics can be applied to create an efficient, intuitive, and easy-to-use pruning algorithm. Our approach achieved excellent results in terms of efficiency and performance, becoming a good alternative for other, more complex, pruning algorithms.

2 Background

This section provides an overview of model pruning and its variants. It also introduces the Lottery Ticket Winner Hypothesis: a key concept in our research.

Model pruning Compression algorithms can be summarized in three areas of research: weight pruning (Han et al., 2015; Zhu and Gupta, 2017), quantization (Gong et al., 2014; Zhu et al., 2016) and knowledge distillation (Ba and Caruana, 2014; Kim and Rush, 2016). These techniques aim to make models lighter, but each of them takes a different approach. Weight pruning removes model parameters according to the chosen algorithm and strategy, while quantization reduces the number of bits necessary to represent each parameter. Knowledge distillation, instead, tries to minimize the learned large knowledge of a model into a smaller one without affecting its *validation*.

Focusing on pruning algorithms, there are different approaches depending on the strategy and algorithm adopted. Pruning algorithms can be classified as either *structured* or *unstructured*, based on the approach applied, and *magnitude-based* or *impact-based*, according to the algorithm used. Structured pruning (Huang et al., 2018; Wang et al., 2019; Gordon et al., 2020) removes weights in groups, such as entire neurons, filters or layers, while unstructured pruning (Han et al., 2015; Frankle and Carbin, 2018; Lagunas et al., 2021; Benbaki et al., 2023) does not consider any relationship between parameters and selects weights to prune based on their impact or magnitude. Magnitude-based algorithms (Hanson and Pratt, 1988; Mozer and Smolensky, 1989; Gordon et al., 2020) analyze the absolute value of each parameter to determine its importance. In contrast, impact-based algorithms (LeCun et al., 1989; Hassibi and Stork, 1992; Singh and Alistarh, 2020) work on the loss function and its variation caused by removing a parameter.

The Lottery Ticket Winner Hypothesis (LTWH: Frankle and Carbin, 2018) is a novel idea related to pruning techniques that proposes the existence of a subnetwork, known as *the winning ticket*, within every neural network that trained in isolation, can yield comparable results to the original model. To identify the winning ticket, a pruning phase is carried out where selected parameters are zero-masked and frozen based on a chosen criterion, such as the smallest magnitude weights. The remaining parameters are then restored to their initial values and re-tuned. The search for the winning ticket can be a one-shot approach or an iterative process where the train-prune-restore steps are repeated multiple times (n times).

3 Related Work

In this section, we discuss two pruning algorithms that are relevant to our research and can be used as comparison benckmarks for KEN. These al-

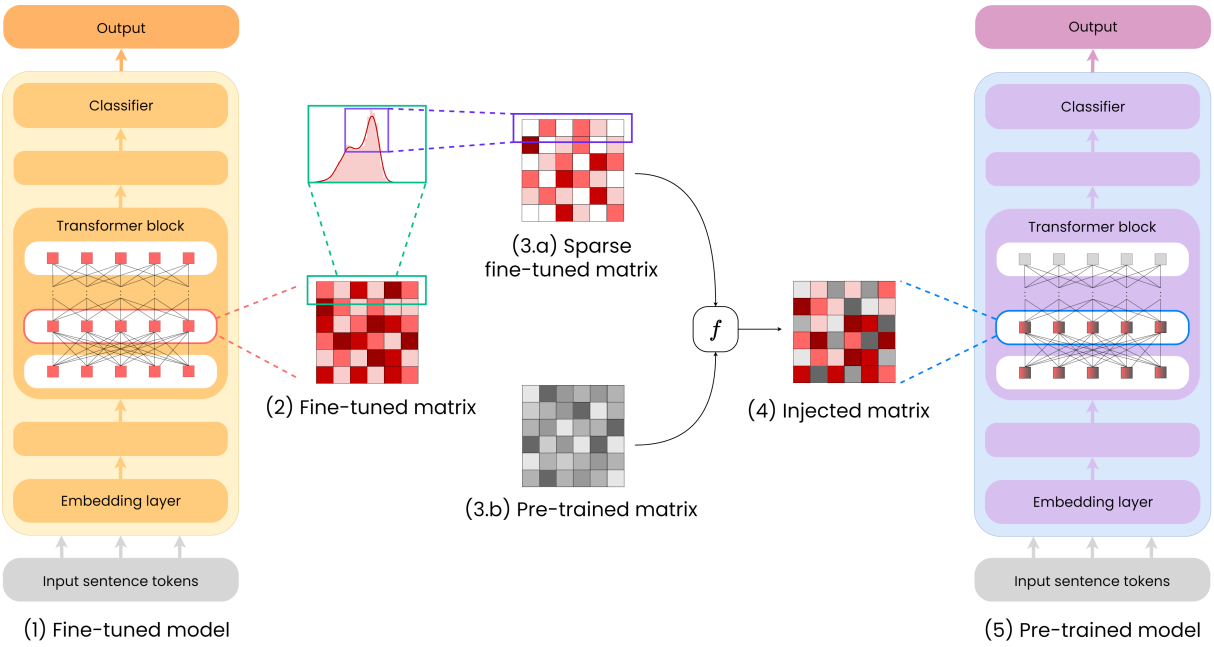


Figure 2: KEN work path: from a fine-tuned model (1), on each of its fine-tuned matrices W^t (2) its sparse version $\tilde{W}$ is calculated (3.a). Using a binary function, the selected values are injected into the pre-trained matrix W^0 (3.b) and the resulting matrix $\hat{W}$ (4) is replaced in a pre-trained model (5)

gorithms, like KEN, are designed to prune transformer models through simple algebraic or geometric techniques. The authors of these papers emphasize the ease of implementation of their respective algorithms.

Factorized Low-rank Pruning (FLOP: Wang et al., 2019) is a simple magnitude-based pruning algorithm that works differently based on the transformer block analyzed: the attention or the embedding block. FLOP parameterizes and factories each matrix W in the attention mechanism into the product of two smaller matrices $PQ : P \in \mathbb{R}^{d' \times r}$ and $Q \in \mathbb{R}^{r \times d}$ with $r \leq \{d, d'\}$. So W represents the sum of r rank-1 components in P and Q. The final step is to apply a diagonal matrix of pruning variables $G = (z_1, ..., z_r)$ obtaining:

$$W = PGQ = \sum_{k=1}^{r} z_k \times (p_k \times q_k)$$

For the embedding layers, instead, FLOP uses an adaptive pruning approach that applies different embedding dimensions and projections to different word clusters. This method prunes most of the dimensions of rare words, which aligns with the empirical choices made in prior work (Joulin et al., 2017; Bastings et al., 2019).

Block Movement Pruning (Lagunas et al., 2021) introduce an extension to the movement pruning technique used in transformers (Sanh et al., 2020). This approach reduces the size of each matrix in a transformer model by dividing it into fixed-sized blocks. Regularization is then applied, and the NN is trained through distillation to match the performance of a teacher model. Our focus is on two pruning methods: Hybrid and HybridNT. The key difference between these two approaches is that HybridNT does not involve the use of a teacher model during training (No Teacher).

4 KEN pruning algorithm

The aim of the KEN (**K**ernel density **E**stimator for **N**eural network compression) pruning algorithm is, based on the main idea of Lottery Ticket Winner Hypothesis (Frankle and Carbin, 2018), to find the optimized fine-tuned subnetwork of each transformer model and inject it into its pre-trained version without affecting its original performance. In addition, this optimized fine-tuned subnetwork can be easily stored in isolation and integrated into the pre-trained model whenever required.

KEN, using the KDE (Kernel Density Estimator) definition, generalizes the point distribution of each transformer matrix and returns its *smooth* and *lightweight* version. Unlike other magnitude-based algorithmic methods, which determine the importance of a parameter by working on both pre-trained and fine-tuned matrices (Ansell et al., 2021),

KEN exclusively focuses on the fine-tuned model. This means that KEN does not extract points based on their importance value. Instead, it aims to identify the subnetwork that generalizes the entire fine-tuned model by taking a sample of all its parameters based on their frequency in each matrix that makes up the model.

To prevent the complete deconstruction of the initial matrix composition, KEN applies KDE on the individual rows of each analyzed matrix. The compression value, denoted by k, determines the number of points used in the KDE calculation. A low value of k implies high compression, which results in a lighter model. Conversely, a high k value leads to low compression, resulting in a model that closely resembles the original version. Fig.1 displays the results of the KDE distribution of a specific matrix in different layers, using the same compression value k.

The KEN algorithm can be described using the three phases defined below:
(Phase 1) Let W^0 the pre-trained matrix of a fixed layer l such that:

$$W^0 = \{w^0_{1,1}, ..., w^0_{n,m}\} \quad | \quad W^0 \in \mathbb{R}^{n \times m}$$

and let W^t the same matrix after the fine-tuning on the task t:

$$W^t = \{w^t_{1,1}, ..., w^t_{n,m}\} \quad | \quad W^t \in \mathbb{R}^{n \times m}$$

For each row r^t_i of the fine-tuned matrix W^t:

$$r^t_i = \{w^t_{i,1}, ..., w^t_{i,m}\} \quad \forall i \in [1, n]$$

the KDE of the row r^t_i is calculated, with a bandwidth

$$h = 1.06 \cdot \hat{\sigma} \cdot n^{-\frac{1}{5}}$$

(Phase 2) Using the KDE likelihood, the k points that best fit the r^t_i row distribution are taken, and the sparse row $\tilde{r}_i$:

$$\tilde{r}_i = \{\tilde{w}_{i,1}, ..., \tilde{w}_{i,m}\} \quad \forall i \in [1, n]$$

is calculated using the following binary function:

$$f(\tilde{w}_{i,j}) = \begin{cases} w^t_{i,j} & \text{if} \quad w^t_{i,j} \in \text{KDE} \quad \text{likelihood} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Combining all the sparse rows, the sparse matrix $\tilde{W}$:

$$\tilde{W} = \{\tilde{w}_{1,1}, ..., \tilde{w}_{n,m}\} \quad | \quad \tilde{W} \in \mathbb{R}^{n \times m}$$

is generated with $k \cdot n$ no-zeros values. $\tilde{W}$ is our *winning ticket* containing the $k \cdot n$ values that will be replaced within the pre-trained matrix W^0.
(Phase 3) The final injected matrix $\hat{W}$:

$$\hat{W} = \{\hat{w}_{1,1}, ..., \hat{w}_{n,m}\} \quad | \quad \hat{W} \in \mathbb{R}^{n \times m}$$

is composed of all the parameters defined using the following function:

$$f(\hat{w}_{i,j}) = \begin{cases} \tilde{w}_{i,j} & \text{if} \quad \tilde{w}_{i,j} \neq 0 \implies \tilde{w}_{i,j} = w^t_{i,j} \\ w^0_{i,j} & \text{otherwise} \end{cases} \tag{2}$$

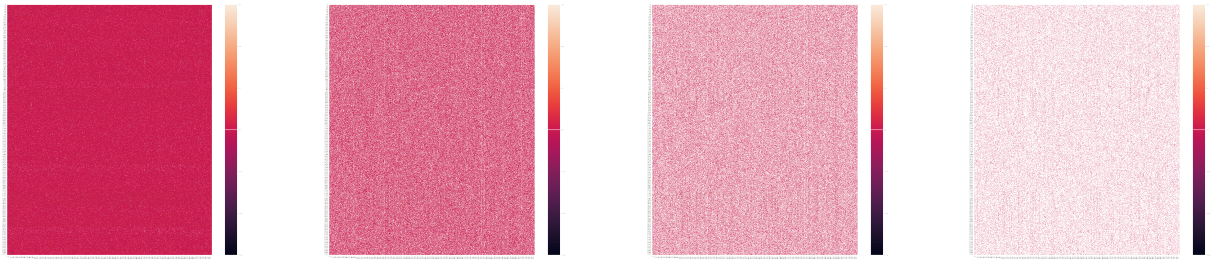
Thus, the injected matrix $\hat{W}$ consists of all the pre-trained W^0 parameters, with the exception of the $k \cdot n$ values in $\tilde{W}$. Once generated, the injected matrix $\hat{W}$ will replace the W^0 matrix in the pre-trained model. Algorithm 1 explains *Phase 1* and *Phase 2*, defining more formally all the steps needed to generate a sparse matrix $\tilde{W}$. Additionally, the graphical representation displayed in Fig.2 provides a clear and comprehensive visualization of all the three phases described while Fig.3 shows different $\tilde{W}$ matrices obtained using different k values.

Algorithm 1: Generate a sparse matrix using KEN

Data: $W^0 = \{w^0_{1,1}, ..., w^0_{n,m}\}$,
$W^t = \{w^t_{1,1}, ..., w^t_{n,m}\}, k$
Result: $\tilde{W}$
$\tilde{W}[n, m] \leftarrow 0$
for *i = 1 to n* **do**
  `best_points` $\leftarrow KDE(r^t_i, k)$
  for *j = 1 to len(r^t_i)* **do**
    $\tilde{r}^t_i \leftarrow []$
    if $r^t_i[j]$ *in* `best_points` **then**
      $\tilde{r}^t_i[j] \leftarrow r^t_i[j]$
    else
      $\tilde{r}^t_i[j] \leftarrow 0$
    end
  end
  $\tilde{W}[i] \leftarrow \tilde{r}^t_i$
end
return $\tilde{W}$

5 Experiments

To validate our algorithm, several case studies were performed. Starting from the main purpose of this work, Sec.5.1 describes the experimental setup, the



(a) Fine-tuned matrix (b) Sparsity: 39.37% (c) Sparsity: 61.42% (d) Sparsity: 83.46%

Figure 3: From the fine-tuned matrix (a), some of its sparse matrices are shown after applying KEN with a different compression value. Matrix (a) is the `in_proj` matrix at layer 0 of a DeBERTa model fine-tuned on `AG_NEWS` dataset. The sparsity percentage refers to the entire model and not to individual matrices. Non-injected values are blank.

models used and the compression value k tested. Furthermore, we also conducted two qualitative analyses to demonstrate the efficacy of KEN. The first analysis sought to establish whether the sparse matrices obtained after the KEN step are indeed the optimal parameters to employ (Sec.5.2) while the second analysis aimed to ascertain the feasibility of saving and loading the compressed data (Sec. 5.3).

5.1 Experimental set-up

The goal of the experiments is to identify the top-performing subnetwork, known as the *winning ticket*, across various models and datasets tested using the KEN algorithm. Training, validation, and test sets were generated for each analyzed dataset if not already provided. This division remained the same for all experiments conducted and for each compression value checked. All the datasets were imported from Huggingface[1]. To ensure high performance, we fine-tuned each dataset with a number of epochs that yielded a fine-tuned model with the highest possible F1-weighted value. Despite what the literature suggests, we used the F1 measure instead of classical accuracy as a comparison metric - if not explicitly used by the comparison benchmarks. This measure delivers more reliable predictions, particularly on strongly unbalanced datasets.

After the training phase, the sparse matrix $\hat{W}$ is generated and injected into each matrix composing the model. The compression value was gradually increased, starting from a compression of 90% until almost the entire original model was reached. Through this increment, it was possible to find the *threshold value*, whereby the compressed model obtained results similar to the fine-tuned model or when the compression value k leads to a catastrophic decline of performances.

For our research, we used five different BERT-like transformer models. To produce a comprehensive analysis, each model has a different architecture, a different attention mechanism or was trained with a different approach. Tab.1 compares the architectures of the models examined, emphasizing the number of layers and the number of parameters of each.

Model		# Layers	# params
Bert	(Devlin et al., 2018)	12	109 M
DistilBERT	(Sanh et al., 2019)	6	66 M
DeBERTa	(He et al., 2020)	12	138 M
Ernie	(Sun et al., 2020)	12	109 M
Electra	(Clark et al., 2020)	12	33 M

Table 1: Properties of the analyzed models

5.2 How to prove the importance of selected parameters

When performing tests, it is essential to determine whether the output matrices represent the best possible compression or if other matrices yield similar results. To answer this question - in parallel with the execution of KEN - new tests were carried out. In these tests, the same number of compression values was used, but parameters were randomly selected to be or not injected into the model. Thus, more formally, for each matrix composing a generic model, the sparse matrix $\tilde{W}$ contains $k \cdot n$ values randomly picked. The results of this analysis are described in Sec. 6.1.

5.3 Model compression

Transformers and other neural network models have large file sizes. A fine-tuned transformer can

[1] https://huggingface.co/datasets

Model	Sparsity (%)	Size	AG-NEWS	EMO	IMDB	YELP_POLARITY	glue-sst2
Bert	47.54	57M	91.6 (±0.7)	**86.0** (±0.5)	84.9 (±0.8)	93.8 (±1.6)	92.8 (±0.5)
	42.29	63M	92.6 (±0.7)	**86.5** (±0.7)	85.3 (±1.7)	94.0 (±1.7)	92.9 (±0.4)
	37.05	69M	**93.4** (±0.1)	84.2 (±1.1)	86.8 (±0.1)	**95.0** (±0.4)	**93.7** (±0.5)
	31.80	75M	**93.7** (±0.2)	**87.4** (±0.7)	**87.3** (±0.1)	**95.0** (±0.5)	**93.7** (±0.4)
	26.55	80M	**93.6** (±0.1)	**87.9** (±0.3)	**87.6** (±0.1)	**95.1** (±0.4)	**93.8** (±0.4)
DistilBERT	45.32	36M	90.4 (±0.2)	**86.7** (±1.6)	78.9 (±2.3)	93.4 (±0.6)	89.0 (±1.3)
	39.86	40M	91.9 (±0.3)	**88.2** (±1.1)	78.1 (±1.4)	94.1 (±0.1)	89.2 (±0.7)
	34.39	44M	92.3 (±0.6)	**88.1** (±1.4)	83.2 (±1.1)	94.6 (±0.1)	**91.9** (±0.2)
	28.92	47M	**93.1** (±0.2)	**88.8** (±0.6)	84.4 (±0.5)	94.7 (±0.1)	**91.9** (±0.1)
	23.45	51M	**93.3** (±0.2)	**88.2** (±0.3)	**84.6** (±0.9)	**94.9** (±0.1)	**92.0** (±0.1)
DeBERTa	44.88	76M	89.4 (±1.2)	83.0 (±4.7)	76.8 (±7.3)	95.5 (±0.4)	94.0 (±0.1)
	39.37	84M	91.4 (±0.6)	**88.9** (±1.5)	82.5 (±3.1)	96.0 (±0.2)	92.8 (±0.4)
	33.86	92M	92.2 (±0.1)	**87.9** (±1.2)	82.5 (±5.1)	95.9 (±0.4)	94.6 (±0.2)
	28.35	99M	**92.7** (±0.1)	**87.3** (±1.0)	88.3 (±1.1)	**96.1** (±0.2)	**94.9** (±0.1)
	22.84	107M	**92.9** (±0.1)	**87.1** (±1.2)	**89.8** (±0.1)	**96.2** (±0.1)	**94.8** (±0.1)
Ernie	47.54	57M	91.5 (±1.4)	**88.3** (±0.4)	87.6 (±0.6)	**95.7** (±0.1)	**94.1** (±0.4)
	42.29	63M	92.7 (±0.6)	**89.1** (±1.1)	89.0 (±0.1)	**95.6** (±0.2)	**93.8** (±0.8)
	37.05	69M	**93.3** (±0.4)	**89.1** (±0.6)	**89.4** (±0.2)	**95.8** (±0.1)	**94.1** (±0.2)
	31.80	75M	**93.3** (±0.3)	**88.7** (±1.2)	**89.2** (±0.2)	**95.8** (±0.3)	**93.8** (±0.2)
	26.55	80M	**93.8** (±0.2)	**88.1** (±0.8)	**89.6** (±0.3)	**95.9** (±0.2)	**93.4** (±0.2)
Electra	73.56	8.9M	84.1 (±2.4)	84.3 (±0.4)	78.9 (±0.5)	88.5 (±0.9)	79.9 (±0.7)
	64.75	12M	89.7 (±0.3)	**86.0** (±0.3)	82.0 (±0.5)	92.1 (±0.8)	85.0 (±0.2)
	55.94	14M	**91.3** (±0.2)	**85.6** (±0.3)	84.3 (±0.1)	93.7 (±0.4)	90.1 (±0.1)

Table 2: Results obtained on various datasets with different sparsity values during the KEN pruning phase. Bold results indicate an equal or better F1-weighted value compared to the original (*unpruned*) model. Other results are shown in Apx.B.

weigh up to 500 MB. However, during the KEN phase, sparse matrices are created and inserted into its pre-trained version. This allows us to create a sparse model consisting solely of all $\tilde{W}$ matrices. To measure the weight reduction achieved by KEN, we can store the compressed model created in this phase and compare it with its original, *unpruned* version. To conduct a thorough analysis, it is crucial to save and load the compressed model accurately. We used the same technique to save both the compressed and the original fine-tuned (unpruned) model to ensure a fair comparison. However, KEN requires a support file, such as a dictionary, to load the compressed model values in the correct position since it injects its weights into a pre-trained model. Sec. 6.2 offers an exhaustive explanation of all the results obtained during this analysis.

6 Results and Discussion

To test the effectiveness of KEN, we conducted several experiments using various classification and sentiment analysis datasets. For each dataset and each compression value, we tested KEN multiple times and calculated the mean and standard deviation of the F1-weighted obtained. The complete list of provided datasets can be found in Apx.A. The results presented in Tab.2 show that KEN can compress all the analyzed models without any impact on their original unpruned performance. KEN achieved a constant compression rate of approximately 25% on all analyzed datasets and 55% on the Electra model. Interestingly, in many cases, there was only a slight difference in performance between models with high sparsity and their counterparts with lower sparsity. This indicates the exceptional generalization capability of KEN even for middle-high compression rates, achieving a good trade-off between performance and compression.

After thoroughly evaluating KEN performance on various datasets, we compared it with other pruning algorithms designed for transformers, such as FLOP, Hybrid and HybridNT described in Sec.3. It is essential to note that Lagunas et al. (2021) models (Hybrid and HybridNT) only prune the attention layers and not the entire model. To facilitate a comprehensive and standardized comparison of all algorithms, we recalibrated the size of their models based on our holistic perspective, ignoring any partial considerations. We presented the results obtained in their publication, adding those obtained by KEN and FLOP in Tab.4. Based on the results obtained, KEN outperforms all other

Model	Pruning algorithm	Size	AG-NEWS	EMO	IMDB	YELP_POLARITY	glue-sst2
Bert	KEN	57M	**91.6** (±0.7)	**86.0** (±0.5)	**84.9** (±0.8)	**93.8** (±1.6)	**92.8** (±0.5)
	Flop	66M	90.9 (±0.9)	83.3 (±0.8)	80.5 (±0.6)	90.2 (±0.6)	83.2 (±0.2)
DistilBERT	KEN	40M	**91.9** (±0.3)	**88.2** (±1.1)	78.1 (±1.4)	**94.1** (±0.1)	**89.2** (±0.7)
	Flop	45M	90.7 (±0.9)	83.2 (±1.2)	**81.2** (±0.9)	90.7 (±0.1)	82.4 (±1.2)
DeBERTa	KEN	84M	**91.4** (±0.6)	**88.9** (±1.5)	**82.5** (±3.1)	**96.0** (±0.2)	**92.8** (±0.4)
	Flop	88M	90.6 (±0.7)	83.1 (±1.7)	81.1 (±0.8)	91.4 (±0.1)	82.3 (±1.1)
Ernie	KEN	57M	**91.5** (±1.4)	**88.3** (±0.4)	**87.6** (±0.6)	**95.7** (±0.1)	**94.1** (±0.4)
	Flop	67M	89.8 (±0.4)	83.8 (±2.3)	81.1 (±0.8)	90.9 (±0.1)	83.2 (±0.9)
Electra	KEN	14M	**91.3** (±0.2)	**85.6** (±0.3)	**84.3** (±0.1)	**93.7** (±0.4)	**90.1** (±0.1)
	Flop	28M	90.9 (±0.3)	83.1 (±2.1)	81.2 (±0.1)	90.5 (±0.1)	81.1 (±0.3)

Table 3: Comparation between KEN and FLOP pruning algorithms on different datasets. Mean and standard deviation are calculated on equal runs for each dataset and algorithm analyzed. Size column indicates the number of injected/compressed parameters used by each algorithm after the pruning phase.

compared models with a significant performance gap, using fewer parameters in all cases.

Model	Size	glue-sst2 Accuracy
Bert-base	109M	93.37
Hybrid	94M	93.23
HybridNT	94M	92.20
KEN	**80M**	**93.80**
Hybrid	66M	91.97
HybridNT	66M	90.71
Sajjad et al. (2020)	66M	90.30
Gordon et al. (2020)	66M	90.80
Flop	66M	83.20
KEN	**63M**	**92.90**

Table 4: Pruning algorithm comparations on SST-2 datasets

In addition to these findings, we conducted a thorough analysis of FLOP, which is the most complete pruning algorithm studied and, like KEN, decomposes original matrices to derive pruned ones. We conducted additional tests on all examined models, using the datasets featured in Tab.2. We compared the results obtained from FLOP to those of KEN, using also in this case, fewer parameters than FLOP. According to the results shown in Tab.3, FLOP performs better than KEN in just one instance. For all other models and datasets analyzed, KEN consistently outperforms FLOP.

These results confirm our hypothesis that integrating fine-tuned sub-networks into pre-trained models is a viable alternative to other pruning techniques.

6.1 Optimal parameters

In order to define the effectiveness of KEN, it is essential to determine whether the parameters introduced by KEN into a generic transformer model constitute the optimal subnetwork or whether the same results can be obtained by randomly selecting the same number of parameters. To investigate this, we conducted an experiment on AG-NEWS dataset. We compared the performance differences between extracting $\tilde{W}$ matrices using KEN and $\tilde{W}$ matrices using $k \cdot n$ random values. The results, illustrated in Fig.4, show that KEN consistently outperforms its random counterpart with a lower error rate and performance gap when using reasonable compression values. It is important to note that in all cases and for all models studied, there is a threshold value beyond which the model performance inevitably suffers a catastrophic decline.

The KEN algorithm can compress a model while maintaining high performance and a minimal error rate. However, if the matrix sparsity exceeds 60%, the performance of the model declines catastrophically. Using random values, this threshold is reached earlier, resulting in a larger performance gap and higher error rate. Nevertheless, the upper bound obtained through this approach is always less than or equal to the mean value obtained through KEN. Furthermore, when using KEN, the error rate is always minimal within the threshold. This indicates that the sub-network obtained using this approach is not random. Instead, it always selects the best possible sub-part of the original network.

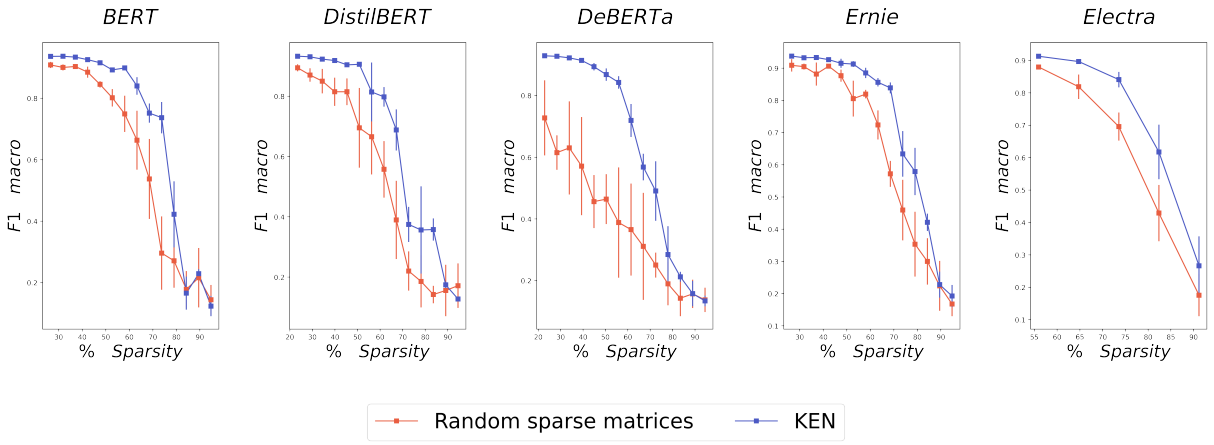


Figure 4: Performance variation on AG-NEWS dataset with different sparsity percentage value.

6.2 Compression values

One main purpose of KEN is to reduce the overall size of transformer models, including their file sizes. To achieve this, KEN leverages sparse matrices, which are created and injected into pre-trained models without affecting their performance. This significantly reduces the final file size by saving only these sparse matrices and injecting them into its pre-trained model. However, a support file (such as a dictionary) is required to inject the values from the compressed model into its pre-trained version. The compressed model is saved using the same techniques and format as the original model, ensuring comparable results. For each model, two compressed versions are obtained using the highest and the lowest sparsity percentages shown in Tab.2.

As shown in Fig.5, both versions of the compressed models result in substantial memory savings, with the size of the compressed models proportional to the sparsity of their matrices. In particular, the compressed model obtained from a low sparsity rate saves about 100 MB on models with original weights up to 400 MB. The support dictionary for parameter injection, held using the Lempel-Ziv-Markov chain data compression algorithm, does not affect the final weight of the model, which is always significantly smaller than the original. Furthermore, the time required to load the injected parameters into the pre-training model is linear with respect to the transformer architecture and the compression performed.

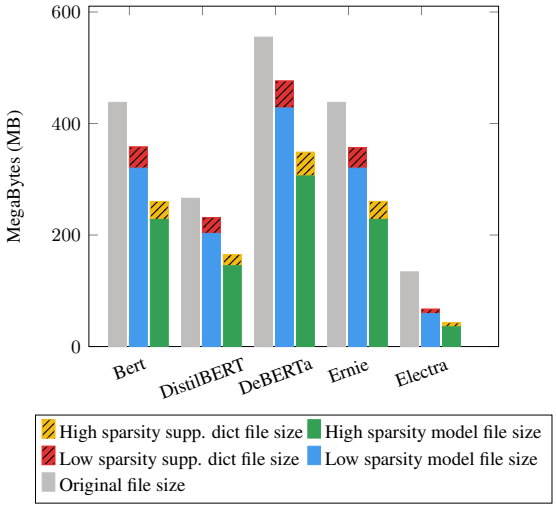


Figure 5: Comparison of the .pt file size between the original and compressed transformer weights

7 Conclusions

Our paper presents KEN: a pruning algorithm that exploits KDE to compress transformer models. KEN works exclusively on the fine-tuned version of the model, returning its lightweight version. We tested KEN on five different transformer models and found that their performances are equal to or better than their unpruned version, with a minimum reduction weight of $\approx 25\%$. We compared KEN with other pruning algorithms and demonstrated that it delivers better results even with fewer parameters. Additionally, it is possible to download the compressed model and inject it into its pre-trained version, resulting in significant memory savings. One of the key strengths of KEN is its simplicity in extracting a compact model and its ability to save the compressed model. Through KEN, we have demonstrated that a simple algorithm can produce excellent results. Moreover, KEN introduces an important step that has not been fully explored in pruning algorithms until now: the ability to load the optimized and compressed model at any time.

8 Limitations

One of the major weaknesses of KEN is its speed, which depends heavily on the size of the matrix being analyzed, particularly the number of rows. KEN can provide excellent results with medium to high k values, resulting in a more detailed distribution and larger point extraction. However, this leads to an increase in computation time, which grows linearly with the size of the matrix, the number of model layers, and the selected k value. It is important to note that this only affects the creation of matrices and not the saving and loading of compressed models.

Moreover, KEN has another limitation, as it is unable to analyze 3D matrices. Since most transformer architectures use 2D matrices, KEN cannot generalize those matrices when analyzing models that work on 3D matrices like XLNet (Yang et al., 2019). Therefore, to fully apply KEN to all transformer models, an extension of the algorithm for 3D matrices is required.

References

Alan Ansell, Edoardo Maria Ponti, Anna Korhonen, and Ivan Vulić. 2021. Composable sparse fine-tuning for cross-lingual transfer. *arXiv preprint arXiv:2110.07560*.

Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep? *Advances in neural information processing systems*, 27.

Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa Anke, and Leonardo Neves. 2020. TweetEval: Unified benchmark and comparative evaluation for tweet classification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1644–1650, Online. Association for Computational Linguistics.

Jasmijn Bastings, Wilker Aziz, and Ivan Titov. 2019. Interpretable neural predictions with differentiable binary variables. *arXiv preprint arXiv:1905.08160*.

Riade Benbaki, Wenyu Chen, Xiang Meng, Hussein Hazimeh, Natalia Ponomareva, Zhe Zhao, and Rahul Mazumder. 2023. Fast as chita: Neural network pruning with combinatorial optimization. *arXiv preprint arXiv:2302.14623*.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146.

Ankush Chatterjee, Kedhar Nath Narahari, Meghana Joshi, and Puneet Agrawal. 2019. SemEval-2019 task 3: EmoContext contextual emotion detection in text. In *Proceedings of the 13th International Workshop on Semantic Evaluation*, pages 39–48, Minneapolis, Minnesota, USA. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

Arman Cohan, Waleed Ammar, Madeleine van Zuylen, and Field Cady. 2019. Structural scaffolds for citation intent classification in scientific publications. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3586–3596, Minneapolis, Minnesota. Association for Computational Linguistics.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. In *Proceedings of the international AAAI conference on web and social media*, volume 11, pages 512–515.

Ona de Gibert, Naiara Perez, Aitor García-Pablos, and Montse Cuadros. 2018. Hate Speech Dataset from a White Supremacy Forum. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 11–20, Brussels, Belgium. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Xin Dong, Shangyu Chen, and Sinno Pan. 2017. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in neural information processing systems*, 30.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*.

Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*.

Antonio Gulli. 2005. Ag's corpus of news articles.

Harsha Gurulingappa, Abdul Mateen Rajput, Angus Roberts, Juliane Fluck, Martin Hofmann-Apitius, and Luca Toldo. 2012. Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. *Journal of Biomedical Informatics*, 45(5):885 – 892.

Text Mining and Natural Language Processing in Pharmacogenomics.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

Stephen Hanson and Lorien Pratt. 1988. Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems*, 1.

Babak Hassibi and David Stork. 1992. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. 2018. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2752–2761.

Steven A Janowsky. 1989. Pruning versus clipping in neural networks. *Physical Review A*, 39(12):6600.

Armand Joulin, Moustapha Cissé, David Grangier, Hervé Jégou, et al. 2017. Efficient softmax approximation for gpus. In *International conference on machine learning*, pages 1302–1310. PMLR.

Phillip Keung, Yichao Lu, György Szarvas, and Noah A. Smith. 2020. The multilingual amazon reviews corpus. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. 2021. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*.

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems*, 2.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. 2020. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pages 6682–6691. PMLR.

Michael C Mozer and Paul Smolensky. 1989. Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *arXiv preprint cs/0506075*.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2020. Poor man's bert: Smaller and faster transformer models. *arXiv preprint arXiv:2004.03844*, 2(2).

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389.

Emily Sheng and David Uthus. 2020. Investigating societal biases in a poetry composition system. In *Proceedings of the Second Workshop on Gender Bias in Natural Language Processing*, pages 93–106, Barcelona, Spain (Online). Association for Computational Linguistics.

Sidak Pal Singh and Dan Alistarh. 2020. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33:18098–18109.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. 2020. Ernie 2.0: A continual pre-training framework for language understanding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8968–8975.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2019. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*.

Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5687–5695.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28.

Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*.

Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.

A List of all analyzed datasets

Tab.5 displays all datasets used to test KEN. The datasets are sorted according to their year of release.

Dataset	Reference
`trec`	Li and Roth, 2002
`AG-NEWS`	Gulli, 2005
`rotten tomatoes`	Pang and Lee, 2005
`IMDB`	Maas et al., 2011
`ade_corpus_v2`	Gurulingappa et al., 2012
`glue-sst2`	Socher et al., 2013
`YELP POLARITY`	Zhang et al., 2015
`hate_speech_offensive`	Davidson et al., 2017
`hate_speech18`	de Gibert et al., 2018
`EMO`	Chatterjee et al., 2019
`scicite`	Cohan et al., 2019
`amazon_reviews_multi`	Keung et al., 2020
`poem sentiment`	Sheng and Uthus, 2020
`tweet_eval-emoji`	Barbieri et al., 2020
`tweet_eval-hate`	Barbieri et al., 2020
`tweet_eval-irony`	Barbieri et al., 2020
`tweet_eval-offensive`	Barbieri et al., 2020
`tweet_eval-feminist`	Barbieri et al., 2020

Table 5: Dataset analyzed

Dataset	Bert	DistilBert	DeBERTa	Ernie	Electra
trec	26.55%	23.45%	22.84%	26.55%	55.94%
rotten Tomatoes	26.55%	34.39%	44.88%	42.29%	55.94%
hate_speech_offensive	26.55%	34.39%	22.84%	26.55%	55.94%
hate_speech18	26.55%	23.45%	33.86%	31.80%	64.75%
scicite	37.05%	28.92%	22.84%	31.80%	55.94%†
ade_corpus_v2	52.78%	45.32%	44.88%	63.28%	73.56%
amazon_reviews_multi	31.80%	34.39%	22.84%	31.80%	55.94%†
poem_sentiment	58.03%	45.32%	22.84%	47.54%	73.56%
tweet_eval-emoji	63.28%	23.45%	44.88%	79.02%	55.94%
tweet_eval-hate	26.55%	61.73%	44.88%	47.54%	55.94%
tweet_eval-irony	26.55%	23.45%	22.84%	26.55%	64.75%
tweet_eval-offensive	26.55%†	34.39%	28.35%	31.80%	55.94%
tweet_eval-femminist	26.55%	39.05%	22.84%	37.05%	64.75%

Table 6: Results obtained from the analysis of additional datasets not shown in Tab.2. The values shown in this table correspond to the minimum compression achieved by KEN without affecting the model performance. The † symbol indicates a compression level below the minimum value reported in Tab.2

B Additional results

In this Appendix, we offer supplementary findings in addition to those shown in Tab.2. These results were obtained using the datasets provided in Appendix A, which were not reported in Tab.2. The results, shown in Tab.6, indicate the sparsity percentage reached by each model analyzed by comparing the F1-weighted obtained with that of its *unpruned* version. We used the same approach as described in Sec.6 and tested each model n times with different k values. However, unlike the results displayed in Tab.2, the sparsity percentage presented in Tab. 6 indicates the first compression values that obtained equal or better results in one or more runs.