Compressing Large-Scale Transformer-Based Models: A Case Study on BERT

Prakhar Ganesh[1*], **Yao Chen**[1*], **Xin Lou**[1], **Mohammad Ali Khan**[1], **Yin Yang**[2], **Hassan Sajjad**[3], **Preslav Nakov**[3], **Deming Chen**[4], **Marianne Winslett**[4]

[1]Advanced Digital Sciences Center, Singapore
[2]College of Science and Engineering, Hamad Bin Khalifa University, Qatar
[3]Qatar Computing Research Institute, Hamad Bin Khalifa University, Qatar
[4]University of Illinois at Urbana-Champaign, USA
{prakhar.g,yao.chen,lou.xin,mohammad.k}@adsc-create.edu.sg,
{yyang,hsajjad,pnakov}@hbku.edu.qa, {dchen,winslett}@illinois.edu

Abstract

Pre-trained Transformer-based models have achieved state-of-the-art performance for various Natural Language Processing (NLP) tasks. However, these models often have billions of parameters, and thus are too resource-hungry and computation-intensive to suit low-capability devices or applications with strict latency requirements. One potential remedy for this is model compression, which has attracted considerable research attention. Here, we summarize the research in compressing Transformers, focusing on the especially popular BERT model. In particular, we survey the state of the art in compression for BERT, we clarify the current best practices for compressing large-scale Transformer models, and we provide insights into the workings of various methods. Our categorization and analysis also shed light on promising future research directions for achieving lightweight, accurate, and generic NLP models.

1 Introduction

Sentiment analysis, paraphrase detection, machine reading comprehension, question answering, text summarization—all these Natural Language Processing (NLP) tasks benefit from pre-training a large-scale generic model on an enormous corpus such as a Wikipedia dump and/or a book collection, and then fine-tuning for specific downstream tasks, as shown in Figure 1. Earlier solutions following this methodology used recurrent neural networks (RNNs) as the base model, for example, ULMFiT (Howard and Ruder, 2018) and ELMo (Peters et al., 2018), but more recent methods use the Transformer architecture (Vaswani et al., 2017), which relies heavily on the attention mechanism.

Popular pre-trained Transformers include BERT (Devlin et al., 2019), GPT-2 (Radford et al., 2019), XLNet (Yang et al., 2019), MegatronLM (Shoeybi et al., 2019), Turing-NLG (Rosset, 2020), T5 (Raffel et al., 2020), and GPT-3 (Brown et al., 2020). These Transformers are—for example, BERT, when first released, improved the state of the art for eleven NLP tasks by sizable margins (Devlin et al., 2019). However, Transformers are also bulky and resource-hungry: For instance, GPT-3 (Brown et al., 2020), a recent large-scale Transformer, has over 175 billion parameters. Models of this size incur high memory consumption, computational overhead, and energy. The problem is exacerbated when we consider devices with lower capacity (e.g., smartphones), and applications with strict latency constraints, (e.g., interactive chatbots).

To put things in perspective, a single training run for GPT-3 (Brown et al., 2020), one of the most powerful and heaviest Transformer-based models, trained on a total of 300 billion tokens, costs well above 12 million USD (Floridi and Chiriatti, 2020). Moreover, fine-tuning or even inference with such a model on a downstream task cannot be done on a GPU with 32GB memory, which is the capacity of Tesla V100, one of the most advanced data center GPUs.

Instead it requires access to high-performance GPU or multi-core CPU clusters, which often means a need to access cloud computing with high computation density, such as the Google Cloud

[*]Both authors contributed equally to this research.

Transactions of the Association for Computational Linguistics, vol. 9, pp. 1061–1080, 2021. https://doi.org/10.1162/tacl_a_00413
Action Editor: Andre Filipe Torres Martins. Submission batch: 4/2021; Revision batch: 6/2021; Published 9/2021.

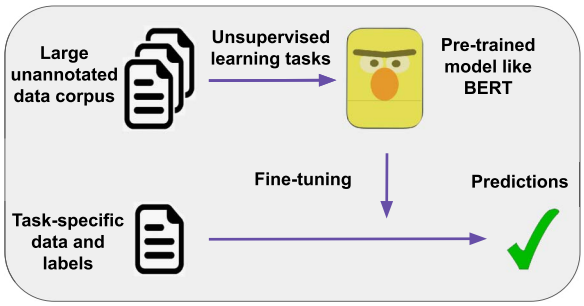

Figure 1: Pre-training large-scale models.

Platform (GCP), Microsoft Azure, Amazon Web Services (AWS), and so forth, and results in a high monetary cost (Floridi and Chiriatti, 2020).

One way to address this problem is through model compression, an intricate part of deep learning that has attracted attention from both researchers and practitioners. A recent study by Li et al. (2020c) highlights the importance of first training over-parameterized models and then compressing them, instead of directly training smaller models, to reduce the performance errors. Although most methods in model compression were originally proposed for convolutional neural networks (CNNs) (pruning, quantization, knowledge distillation, etc.) (Cheng et al., 2018), many ideas are directly applicable to Transformers. There are also methods designed specifically for Transformers (e.g., attention head pruning, attention decomposition, replacing Transformer blocks with an RNN or a CNN), which we will discuss in Section 3. Unlike CNNs, a Transformer model has a relatively complex architecture consisting of multiple parts such as embedding layers, self-attention, and feed-forward layers (details introduced in Section 2). Thus, the effectiveness of different compression methods can vary when applied to different parts of a Transformer model.

Several recent surveys have focused on pre-trained representations and large-scale Transformer-based models (Qiu et al., 2020; Rogers et al., 2020; Wang et al., 2020a). However, to the best of our knowledge, no comprehensive, systematic study has compared the effectiveness of different model compression techniques on Transformer-based large-scale NLP models, even though a variety of approaches for compressing such models have been proposed. Motivated by this, here we offer a thorough and in-depth comparative study on compressing Transformer-based NLP models, with a special focus on the widely used BERT (Devlin et al., 2019). Although the compression methods discussed here can be extended to Transformer-based decoders and multilingual Transformer models, we restrict our discussion to BERT in order to be able to provide more detailed insights into the various methods that we compare.

Our study is timely, since (*i*) the use of Transformer-based BERT-like models has grown dramatically, as demonstrated by current leaders of various NLP tasks such as language understanding (Wang et al., 2018), machine reading comprehension (Rajpurkar et al., 2016, 2018), machine translation (Machacek and Bojar, 2014), summarization (Narayan et al., 2018), and so on; (*ii*) many researchers are left behind as they do not have expensive GPUs (or a multi-GPU setup) with a large amount of GPU memory, and thus cannot fine-tune and use the large BERT model for relevant downstream tasks; and (*iii*) AI-powered devices such as smartphones would benefit tremendously from an on-board BERT-like model, but do not have the capability to run it. In addition to summarizing existing techniques and best practices for BERT compression, we point out several promising future directions of research for compressing large-scale Transformer-based models.

2 Breakdown and Analysis of BERT

Bidirectional Encoder Representations from Transformers, or BERT (Devlin et al., 2019), is a Transformer-based model (Vaswani et al., 2017) pre-trained on large corpora from Wikipedia and the Bookcorpus dataset (Zhu et al., 2015) using two training objectives: (*i*) Masked Language Model (MLM), which helps it learn the context in a sentence, and (*ii*) Next Sentence Prediction (NSP), from which it learns the relationship between two sentences. Subsequent Transformer architectures have further improved the training objective in various ways (Lan et al., 2020; Liu et al., 2019b). In the following, we focus on the original BERT model.

BERT decomposes the input sentence(s) into WordPiece tokens (Wu et al., 2016). Specifically, WordPiece tokenization helps improve the representation of the input vocabulary and reduce its size, by segmenting complex words into subwords. These subwords can even form new words not seen in the training samples, thus making the

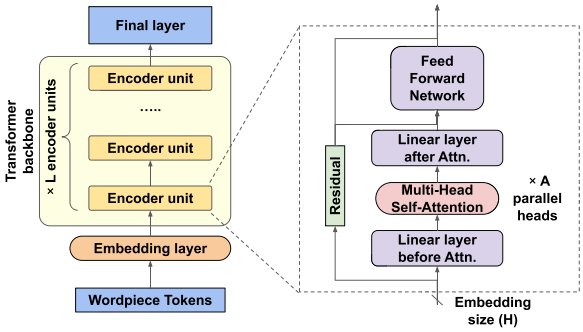

Figure 2: BERT model flowchart.

model more robust to out-of-vocabulary (OOV) words. BERT further inserts a classification token ([CLS]) before the input tokens, and the output corresponding to this token is used for classification tasks that target the entire input. For sentence pair tasks, the two sentences are packed together by inserting a further separator token ([SEP]) between them.

BERT represents each WordPiece token with three vectors, namely, its token, segment, and position embeddings. These embeddings are summed together and then passed through the main body of the model (i.e., the Transformer backbone), which produces the output representations that are fed into the final, application-dependent layer (e.g., a classifier for sentiment analysis).

As shown in Figure 2, the Transformer backbone consists of multiple stacked encoder units, each with two major sub-units: a self-attention sub-unit and a feed forward network (FFN) sub-unit, both with residual connections. Each self-attention sub-unit consists of a multi-head self-attention layer, and fully connected layers before and after it. An FFN sub-unit exclusively contains fully connected layers. The architecture of BERT can be specified using the following three hyper-parameters: number of encoder units (L), size of the embedding vector (H), and number of attention heads in each self-attention layer (A). L and H determine the depth and the width of the model, whereas A is an internal hyper-parameter that affects the number of contextual relations that each encoder can focus on. The authors of BERT provided two pre-trained models:

- $\text{BERT}_{\text{BASE}}$ ($L = 12; H = 768; A = 12$);
- $\text{BERT}_{\text{LARGE}}$ ($L = 24; H = 1024; A = 16$).

We conducted various experiments with the $\text{BERT}_{\text{BASE}}$ model by running inference on a sentence of length 256, and then we collected the results in Figure 3. The top graph in the figure compares the model size as well as the theoretical computational requirements (measured in millions of FLOPs) of different parts of the model. The bottom two graphs track the model's run-time memory consumption as well as the inference latency on two representative hardware setups.

We conducted our experiments using Nvidia Titan X GPU with 12GB of video RAM and Intel Xeon E5-1620 CPU with 32 GB of system memory, which is a commonly used server or workstation configuration. All data was collected using the PyTorch profiling tool.

Clearly, the parts consuming the most memory in terms of model size and executing the highest number of FLOPs are the FFN sub-units. The embedding layer is also a substantial part of the model size, due to the large vector size (H) used to represent each embedding vector. Note that it has zero FLOPs, since it is a lookup table that involves no arithmetic computations at inference time. For the self-attention sub-units, we further break down the costs into multi-head self-attention layers and the linear (i.e., fully connected) layers before and after them. The multi-head self-attention does not have any learnable parameters; however, its computational cost is non-zero due to the dot products and the softmax operations.

The linear layers surrounding each attention layer incur additional memory and computational overhead, though it is relatively small compared to the FFN sub-units. Note that the input to the attention layer is divided among various heads, and thus each head operates in a lower-dimensional space (H/A). The linear layer before attention is roughly three times the size of that after it, since each attention has three inputs (key, value, and query) and only one output.

The theoretical computational overhead may differ from the actual inference cost at run-time, which depends on the hardware that the model runs on. As expected, when running the model on a GPU, the total run-time memory includes memory both on the GPU side and on the CPU side, and it is greater than for a model running solely on a CPU due to duplicate tensors present on both devices for faster processing on a GPU.

The most notable difference between the theoretical analysis and the run-time measurements on a GPU is that the multi-head self-attention layers are significantly more costly in practice than in

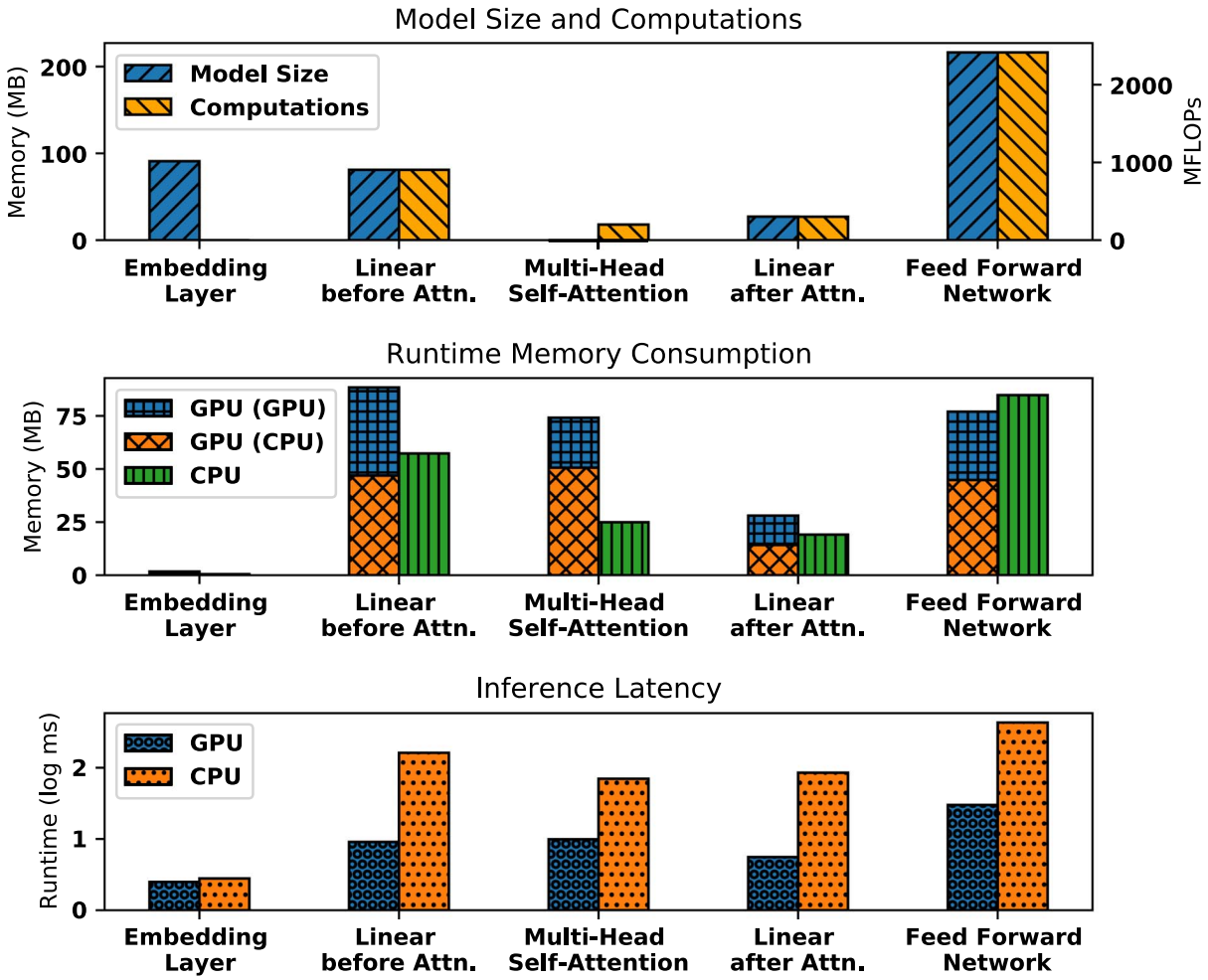

Figure 3: Breakdown analysis of $\text{BERT}_{\text{BASE}}$.

theory. This is because the operations in these layers are complex, and are implemented as several matrix transformations followed by a matrix multiplication and a softmax. Moreover, GPUs are designed to accelerate certain operations, and thus can implement linear layers faster and more efficiently than the more complex attention layers.

When we compare the run-time performance on a CPU, where the hardware is not specialized for linear layer operations, the inference time as well as the memory consumption of all the linear layers shoots up more compared to the multi-head self-attention. Thus, on a CPU, the behavior of run-time performance is similar to that of theoretical computations. The total execution time for a single example on a GPU (57.1 ms) is far superior as compared to a CPU (750.9 ms), as expected. The execution time of the embedding layer is largely independent of the hardware on which the model is executed (since it is just a table lookup) and it is relatively small compared to the other layers. The FFN sub-units are the bottleneck of the whole model, which is consistent with the results from the theoretical analysis.

3 Compression Methods

Because of BERT's complex architecture, no existing compression method has focused on every single aspect of the model: self-attention, linear layers, embedding size, model depth, and so forth. Instead, each compression method applies to certain components of BERT. Below, we consider the compression methods that offer model size reduction and speedup at inference time, rather than during the training procedure.

3.1 Quantization

Quantization refers to reducing the number of unique values required to represent model weights and activations, which allows to represent them

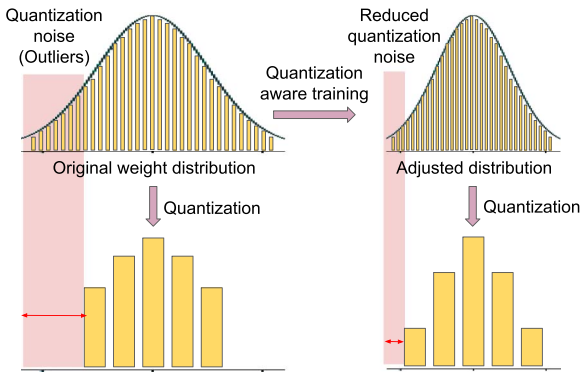

Figure 4: Quantization.

using fewer bits, to reduce the memory footprint, and to lower the precision of the numerical calculations. Quantization may even improve the runtime memory consumption as well as the inference speed when the underlying computational device is optimized to process lower-precision numerical values, for example, tensor cores in newer generations of Nvidia GPUs. Programmable hardware such as FPGAs can also be specifically optimized for any bitwidth representation. Quantization of intermediate outputs and activations can further speed up the model execution (Boo and Sung, 2020).

Quantization is applicable to all model weights as the BERT weights reside in fully connected layers (i.e., the embedding layer, the linear layers, and the FFN sub-units), which have been shown to be quantization-friendly (Hubara et al., 2017). The original BERT model provided by Google represents each weight by a 32-bit floating point number. A naïve approach is to simply truncate each weight to the target bitwidth, which often yields a sizable drop in accuracy as this forces certain weights to go through a severe drift in their value, known as quantization noise (Fan et al., 2021).

A possible way around this issue is to identify these weights and then not to truncate them during the quantization step in order to retain the model accuracy. For example, Zadeh et al. (2020) assumed Gaussian distribution in the weight matrix and identified the outliers. Then, by not quantizing these outliers, they were able to perform post-training quantization without any retraining requirements.

A more common approach to retaining the model accuracy is Quantization-Aware Training (QAT), which involves additional training steps to adjust the quantized weights. Figure 4 shows an example of naïve linear quantization, quantization noise, and the importance of quantization-aware training. For BERT, QAT has been used to perform fixed-length integer quantization (Zafrir et al., 2019; Boo and Sung, 2020), Hessian-based mixed-precision quantization (Shen et al., 2020), adaptive floating-point quantization (Tambe et al., 2020), and noise-based quantization (Fan et al., 2021). Finally, it has been observed that the embedding layer is more sensitive to quantization than the other encoder layers, and thus that it requires more bits in order to maintain the model accuracy (Shen et al., 2020).

3.2 Pruning

Pruning refers to identifying and removing redundant or less important weights and/or components, which sometimes even makes the model more robust and better-performing. Moreover, pruning is a commonly used method of exploring the lottery ticket hypothesis in neural networks (Frankle and Carbin, 2019), which has also been studied in the context of BERT (Chen et al., 2020b; Prasanna et al., 2020). Pruning methods for BERT largely fall into two categories, which we explore below.

Unstructured Pruning. Unstructured pruning, also known as sparse pruning, prunes individual weights by locating the set of the least important weights in the model. The importance of the weights can be judged by their absolute values, by the gradients, or by some custom-designed measurement (Gordon et al., 2020; Mao et al., 2020; Guo et al., 2019; Sanh et al., 2020; Chen et al., 2020b). Unstructured pruning could be effective for BERT, given the latter's massive amount of fully-connected layers. Unstructured pruning methods include magnitude weight pruning (Gordon et al., 2020; Mao et al., 2020; Chen et al., 2020b), which simply removes weights that are close to zero, movement-based pruning (Sanh et al., 2020; Tambe et al., 2020), which removes weights that move towards zero during fine-tuning, and reweighted proximal pruning (RPP) (Guo et al., 2019), which uses iteratively reweighted ℓ_1 minimization followed by the proximal algorithm for decoupling pruning and error back-propagation. Since unstructured pruning considers each weight individually, the set of pruned weights can be arbitrary and irregular, which in turn might decrease the model size, but

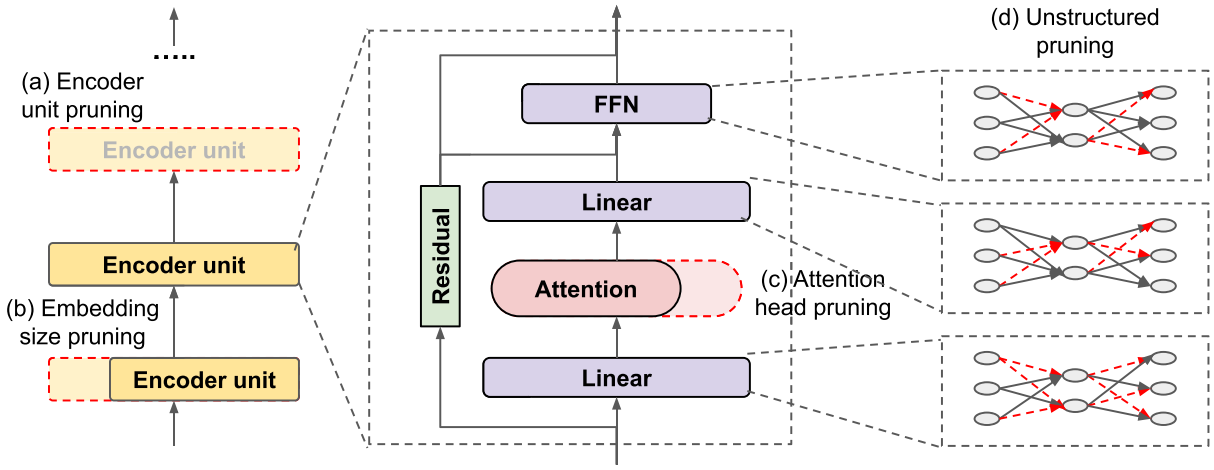

Figure 5: Various pruning methods including structured pruning by (a) pruning number of encoder units (L), (b) pruning embedding size (H), (c) pruning number of attention heads (A), as well as (d) unstructured pruning.

with negligible improvement in runtime memory or speed, unless executed on specialized hardware or with specialized processing libraries.

Structured Pruning. Unlike unstructured pruning, structured pruning focuses on pruning structured blocks of weights (Li et al., 2020a) or even complete architectural components in the BERT model, by reducing and simplifying certain numerical modules:

- *Attention Head Pruning.* As we have seen above, the self-attention layer incurs considerable computational overhead at inference time; yet, its importance has often been questioned (Kovaleva et al., 2019; Tay et al., 2020; Raganato et al., 2020). In fact, it has been shown that high accuracy is possible with only 1–2 attention heads per encoder unit, even though the original BERT model had 16 attention heads (Michel et al., 2019). Randomly pruning attention heads during the training phase has also been proposed, which can create a model that is robust to various numbers of attention heads, and thus a smaller model can be directly extracted for inference based on the required deployment requirements (Hou et al., 2020).

- *Encoder Unit Pruning.* Another structured pruning method aims to reduce the number of encoder units L by pruning the less important layers. For instance, layer dropout drops encoder units randomly or with a pre-defined strategy during training. If the layers are dropped randomly, a smaller model of any desired depth can be extracted during inference (Fan et al., 2020; Hou et al., 2020). Otherwise, a smaller model of fixed depth is obtained (Sajjad et al., 2020; Xu et al., 2020). As BERT contains residual connections for every sub-unit, using an identity prior to prune these layers has also been proposed (Lin et al., 2020).

- *Embedding Size Pruning.* Similarly to encoder unit pruning, we can reduce the size of the embedding vector (H) by pruning along the width of the model. Such a model can be obtained by either training with adaptive width, so that the model is robust to such pruning during inference (Hou et al., 2020), or by removing the least important feature dimensions iteratively (Khetan and Karnin, 2020; Prasanna et al., 2020; Tsai et al., 2020; Lin et al., 2020).

Figure 5 shows a visualization of various forms of structured pruning and unstructured pruning.

3.3 Knowledge Distillation

Knowledge Distillation refers to training a smaller model (called the *student*) using outputs (from various intermediate functional components) of one or more larger pre-trained models (called the *teachers*). The flow of information can sometimes be through an intermediate model (commonly known as *teaching assistants*) (Ding and Yang, 2020; Sun et al., 2020b; Wang et al., 2020c). In the BERT model, there are multiple intermediate results that the student can potentially

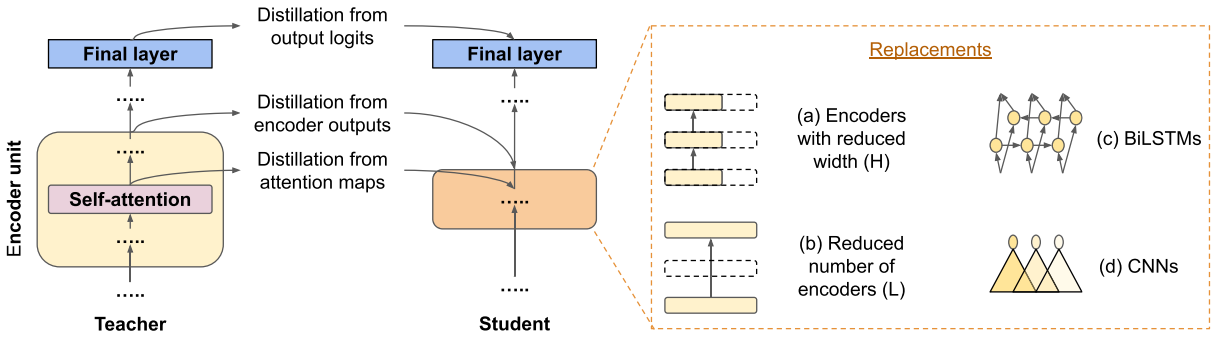

Figure 6: Knowledge distillation. Student models can be formed by (a) reducing the encoder width, (b) reducing the number of encoders, (c) replacing with a BiLSTM, (d) replacing with a CNN, or some combination thereof.

learn from, such as the logits in the final layer, the outputs of the encoder units, and the attention maps. Moreover, there are multiple forms of loss functions that can be adapted for this purpose such as cross-entropy loss, KL divergence, MAE, and so on. While knowledge distillation is most commonly used to train student models directly on task-specific data, recent results have shown that distillation during both pre-training and fine-tuning can help create better performing models (Song et al., 2020). An overview of various forms of knowledge distillation and student models is shown in Figure 6. Based on what the student learns from the teacher, we can categorize the existing methods as follows:

Distillation from Output Logits. Similar to knowledge distillation for CNNs (Cheng et al., 2018), the student can directly learn from the output logits (i.e., from soft labels) of the final softmax layer in BERT. This is done to allow the student to better mimic the output of the teacher model, by replicating the probability distribution across various classes.

While knowledge distillation on output logits is most commonly used to train smaller BERT models (Sun et al., 2019; Sanh et al., 2019; Jiao et al., 2020; Zhao et al., 2019b; Cao et al., 2020; Sun et al., 2020b; Song et al., 2020; Mao et al., 2020; Li et al., 2020b; Ding and Yang, 2020; Noach and Goldberg, 2020), the student does not need to be a smaller version of BERT or even a Transformer, and can follow a completely different architecture. Below we describe the two commonly used replacements:

- *Replacing the Transformer with a BiLSTM*, to create a lighter backbone. Recurrent models such as BiLSTMs process words sequentially instead of simultaneously attending to each word in the sentence like Transformers do, resulting in a smaller runtime memory requirement. Both can create bidirectional representations, and thus BiLSTMs can be considered a faster alternative to Transformers (Wasserblat et al., 2020). Compressing to a BiLSTM is typically done directly for a specific NLP task (Mukherjee and Awadallah, 2020). Since these models are trained from scratch on the task-specific dataset without any intermediate guidance, various methods have been proposed to create additional synthetic training data using rule-based data augmentation techniques (Tang et al., 2019; Mukherjee and Awadallah, 2020) or to collect data from multiple tasks to train a single model (Liu et al., 2019a).

- *Replacing the Transformer with a CNN*, to take advantage of massively parallel computations and improved inference speed (Chia et al., 2018). While it is theoretically possible to make the internal processing of an encoder parallel, where each parallel unit requires access to all the inputs from the previous layer as an encoder unit focuses on the global context, this setup is computationally intensive and cost-inefficient. Unlike Transformers, each CNN unit focuses on local context, and, unlike BiLSTMs, CNNs do not operate on the input sequentially, which makes it easier for them to divide the computation into small parallel units. It is possible to either completely replace the Transformer backbone with a deep CNN network (Chen et al., 2020a), or to replace only a few encoder units to balance performance and efficiency (Tian et al., 2019).

Distillation from Encoder Outputs. Each encoder unit in a Transformer model can be viewed as a separate functional unit. Intuitively, the output tensors of such an encoder unit may contain meaningful semantic and contextual relationships between input tokens, leading to an improved representation. Following this idea, we can create a smaller model by learning from an encoder's outputs. The smaller model can have a reduced embedding size H, a smaller number of encoder units L, or a lighter alternative that replaces the Transformer backbone.

- Reducing the number of heads H yields more compact representations in the student (Zhao et al., 2019b; Sun et al., 2020b; Jiao et al., 2020; Li et al., 2020b). One challenge is that the student cannot directly learn from the teacher's intermediate outputs, due to different sizes. To overcome this, the student also learns a transformation, which can be implemented by either down-projecting the teacher's outputs to a lower dimension or by up-projecting the student's outputs to the original dimension (Zhao et al., 2019b). Another possibility is to introduce these transformations directly into the student model, and later to merge them with the existing linear layers to obtain the final smaller model (Zhou et al., 2020a).

- Reducing the number of encoder units L forces each encoder unit in the student to learn from the behavior of a sequence of multiple encoder units in the teacher (Sun et al., 2019; Sanh et al., 2019; Sun et al., 2020b; Jiao et al., 2020; Zhao et al., 2019b; Li et al., 2020b). Further analysis into various details of choosing which encoder units to use for distillation is provided by Sajjad et al. (2020). For example, preserving the bottom encoder units and aggressively distilling the top encoder units yields a better-performing student model, which indicates the importance of the bottom layers in the teacher model. While most existing methods create an injective mapping from the student encoder units to the teacher, Li et al. (2020b) instead proposed a way to build a many-to-many mapping for a better flow of information. One can also completely bypass the mapping by combining all outputs into one single representation vector (Sun et al., 2020a).

- It is also possible to use encoder outputs to train student models that are not Transformers (Mukherjee and Awadallah, 2020; Tian et al., 2019). However, when the student model uses a completely different architecture, the flexibility of using internal representations is rather limited, and only the output from the last encoder unit can be used for distillation.

Distillation from Attention Maps. An attention map refers to the softmax distribution output of the self-attention layers and indicates the contextual dependence between the input tokens. It has been proposed that attention maps in BERT can identify distinguishable linguistic relations, for example, identical words across sentences, verbs and corresponding objects, or pronouns and corresponding nouns (Clark et al., 2019). These distributions are the only source of inter-dependency between input tokens in a Transformer model, and thus by replicating these distributions, a student can also learn such linguistic relations (Sun et al., 2020b; Jiao et al., 2020; Mao et al., 2020; Tian et al., 2019; Li et al., 2020b; Noach and Goldberg, 2020).

A common method of distillation from attention maps is to directly minimize the difference between the teacher's and the student's multi-head self-attention outputs. Similarly to distillation from encoder outputs, replicating attention maps also faces a choice of mapping between the teacher and the student, as each encoder unit has its own attention distribution. Previous work has also proposed replicating only the last attention map in the model to truly capture the contextual dependence (Wang et al., 2020c). One can attempt an even deeper distillation of information through intermediate attention outputs such as key, query, and value matrices, individual attention head outputs, key–query, and value–value matrix products, and so forth, to facilitate the flow of information (Wang et al., 2020c; Noach and Goldberg, 2020).

3.4 Matrix Decomposition

The computational overhead in BERT mainly consists of large matrix multiplications, both in the linear layers and in the attention heads. Thus, decomposing these matrices can significantly impact the computational requirements for such models.

Weight Matrix Decomposition. The computational overhead of the model can be reduced

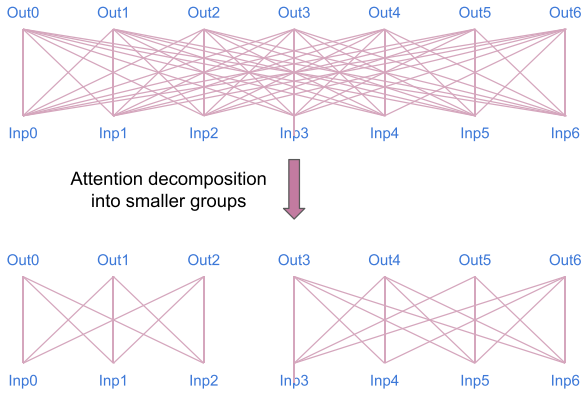

Figure 7: Attention decomposition.

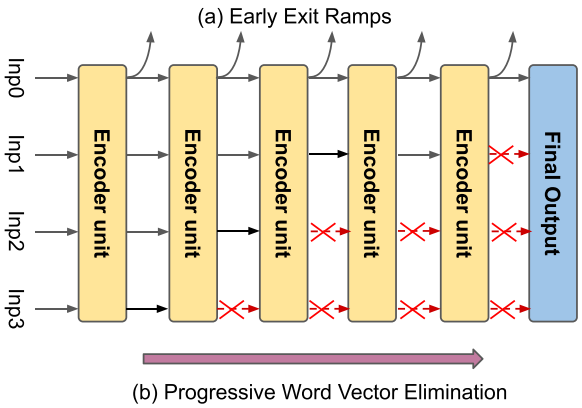

Figure 8: Dynamic inference acceleration.

through weight matrix factorization, which replaces the original $A \times B$ weight matrix by the product of two smaller ones ($A \times C$ and $C \times B$). The reduction in model size and runtime memory use is sizable if $C \ll A, B$. The method can be applied to the linear layers (Noach and Goldberg, 2020; Mao et al., 2020), or to the embedding matrix (Lan et al., 2020; Tambe et al., 2020).

Attention Decomposition. It has been shown that computing attention over the entire sentence makes a large number of redundant computations (Tay et al., 2020; Cao et al., 2020). Thus, it has been proposed to do it in smaller groups, by either binning them using spatial locality (Cao et al., 2020), magnitude-based locality (Kitaev et al., 2020), or an adaptive attention span (Tambe et al., 2020). Moreover, since the outputs are calculated independently, local attention methods also enable a higher degree of parallel processing and individual representations can be saved during inference for multiple uses. Figure 7 shows an example of attention decomposition based on spatial locality.

It has been also proposed to reduce the attention computations by projecting the key–query matrix into a lower dimensionality (Wang et al., 2020b) or by only calculating the softmax of the top-k key-query product values in order to further highlight these relations (Zhao et al., 2019a).

Since the multi-head self-attention layer contains no weights, these methods only improve the runtime memory costs and execution speed, but do not reduce the model size.

3.5 Dynamic Inference Acceleration

Besides directly compressing the model, some methods focus on reducing the computational overhead at inference time by catering to individual input examples and dynamically changing the amount of computation. Figure 8 shows a visualization of two such methods, which we discuss below.

Early Exit Ramps. One way to speed up inference is to create intermediary exit points in the model. Since the classification layers are the least parameter-extensive part of BERT, separate classifiers can be trained for each encoder unit output. This allows the model to have dynamic inference time for various inputs. Training these separate classifiers can be done either from scratch (Xin et al., 2020; Zhou et al., 2020b; Tambe et al., 2020) or by distilling the output of the final classifier (Liu et al., 2020).

Progressive Word Vector Elimination. Another way to accelerate inference is by reducing the number of words processed at each encoder level. Since we only use the final output corresponding to the [CLS] token (defined in Section 2) as a representation of the complete sentence, the information of the entire sentence must have fused into that one token. Goyal et al. (2020) observed that such a fusion cannot be sudden, and that it must happen progressively across various encoder levels. We can use this information to lighten the later encoder units by reducing the sentence length through word vector elimination at each step.

3.6 Other Methods

Besides the aforementioned methods, there are also several one-of-a-kind methods that have been shown to be effective for reducing the size and the inference time of BERT-like models.

Parameter Sharing. ALBERT (Lan et al., 2020) uses the same architecture as BERT, but with weights shared across all encoder units, which reduces memory consumption significantly. Moreover, ALBERT enables training larger and deeper models: While BERT's performance peaks at $\text{BERT}_{\text{LARGE}}$ (performance of $\text{BERT}_{\text{XLARGE}}$ drops significantly), ALBERT keeps improving until the far larger $\text{ALBERT}_{\text{XXLARGE}}$ model ($L = 12; H = 4096; A = 64$).

Embedding Matrix Compression. The embedding matrix is the lookup table for the embedding layer, which is about 21% of the size of the complete BERT model. One way to compress it is by reducing the vocabulary size V, which is about 30k in the original BERT model. Recall from Section 2 that the vocabulary of BERT is learned using a WordPiece tokenizer, which relies on the vocabulary size to figure out the degree of fragmentation of the words in the input text. A large vocabulary size allows for better representation of rare words and for more adaptability to out-of-vocabulary words. However, even with a 5k vocabulary size, 94% of the tokens match those created using a 30k vocabulary size (Zhao et al., 2019b). Thus, the majority of the words that appear frequently enough are covered even with a small vocabulary size, which makes it reasonable to decrease the vocabulary size to compress the embedding matrix. Another alternative is to replace the existing one-hot vector encoding with a ''codebook''-based one, where each token is represented using multiple indices from the codebook. The final embedding of the token can then be calculated as the sum of the embeddings present in all these indices (Prakash et al., 2020).

Weight Squeezing. Weight squeezing (Chumachenko et al., 2020) is a compression method similar to knowledge distillation, where the student learns from the teacher. However, instead of learning from intermediate outputs as in knowledge distillation, the weights of the teacher model are mapped to the student through a learnable transformation, and thus the student learns its weights directly from the teacher.

4 Effectiveness of Compression Methods

In this section, we compare the performance of several BERT compression techniques based on their model size and speedup, as well as their accuracy or F1 score on various NLP tasks. We chose work whose results are either on the Pareto frontier (Deb, 2014) or representative for each compression technique mentioned in the previous section.

4.1 Datasets and Evaluation Measures

From the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) and the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016), we use the following most common tasks: MNLI and QQP for sentence pair classification, SST-2 for single sentence classification, and SQuAD v1.1 for machine reading comprehension. Following the official leaderboards, we report the accuracy for MNLI, SST-2, and QQP, and F1 score for SQuAD v1.1. In an attempt to quantify the results on a single scale, we also report the absolute drop in performance with respect to $\text{BERT}_{\text{BASE}}$, averaged across all tasks for which the authors have reported results.

We further report speedup on both GPU and CPU devices, collected directly from the original papers. For papers that report speedup, we also mention the target device on which is was calculated, and for such that do not, we run their models on our own machine and we perform inference on the complete MNLI-m test set (using a batch size of 1) with machine configurations as detailed in Section 2. We also report the model size with and without the embedding matrix, since for certain application scenarios, where the memory constraints for model storage are not strict, the parameters of the embedding matrix can be ignored as it has negligible run-time cost (see Section 2). As no previous work has reported the drop in runtime memory, and as many papers that we compare to use probabilistic models that cannot be easily replicated without their code, we could not perform direct runtime memory comparisons.

4.2 Comparison and Analysis

Table 1 compares various BERT compression methods. While some compress only part of the model, for uniformity, we report size and speedup for the final complete models after compression. Thus, certain values might not match exactly what is reported in the original papers.

Methods	Provenance	Target	Model Size		Speedup		Accuracy/F1				Avr.
		Device	w/ emb	w/o emb	GPU	CPU	MNLI	QQP	SST-2	SQD	Drop
$\text{BERT}_{\text{BASE}}$	(Devlin et al., 2019)	–	100%	100%	1x	1x	84.6	89.2	93.5	88.5	0.0
Quantization	(Shen et al., 2020) ^{S}	–	15%	12.5%	1x	1x	83.9	–	92.6	88.3	−0.6
	(Zadeh et al., 2020) ^{S}	–	10.2%	5.5%	1x	1x	83.7	–	–	–	−0.9
Unstructured Pruning	(Guo et al., 2019) ^{A}	–	67.6%	58.7%	1x	1x	–	–	–	88.5	0.0
	(Chen et al., 2020b) ^{S}	–	48.9%*	35.1%*	1x	1x	83.1	89.5	92.9	87.8	−0.63
	(Sanh et al., 2020) ^{S}	–	23.8%	3%	1x	1x	79.0	89.3	–	79.9	−4.73
Structured Pruning	(Lin et al., 2020) ^{S}	–	60.7%	50%	–	–	–	88.9	91.8	–	−1.0
	(Khetan and Karnin, 2020) ^{A}	–	39.1%	38.8%	2.93x‡	2.76x‡	83.4	–	90.9	86.7	−1.86
KD from Output Logits	(Song et al., 2020) ^{A,S}	V100	22.8%	10.9%	6.25x	7.09x	–	88.6	92.9	–	−0.6
	(Liu et al., 2019a)† ^{S}	V100	24.1%	3.3%	10.7x	8.6x‡	78.6	88.6	91.0	–	−3.03
	(Chen et al., 2020a) ^{A,S}	V100	7.4%	4.8%	19.5x*	–	81.6	88.7	91.8	–	−2.06
KD from Attn.	(Wang et al., 2020c) ^{A}	P100	60.7%	50%	1.94x	1.73x	84.0	91.0	92.0	–	−0.1
Multiple KD combined	(Sanh et al., 2019) ^{A}	CPU	60.7%	50%	1.94x	1.73x	82.2	88.5	91.3	86.9	−1.73
	(Sun et al., 2020b)† ^{A}	Pixel	23.1%	24.8%	3.9x‡	4.7x‡	83.3	–	92.8	90.0	−0.16
	(Jiao et al., 2020) ^{A,S}	K80	13.3%	6.4%	9.4x	9.3x‡	82.5	89.2	92.6	–	−1.0
	(Zhao et al., 2019b) ^{A}	–	1.6%	1.8%	25.5x‡	22.7x‡	71.3	–	82.2	–	−12.3
Matrix Decomposition	(Noach and Goldberg, 2020) ^{S}	Titan V	60.6%	49.1%	0.92x	1.05x	84.8	89.7	92.4	–	−0.13
	(Cao et al., 2020) ^{S}	V100	100%	100%	3.14x	3.55x	82.6	90.3	–	87.1	−0.76
Dynamic Inference	(Xin et al., 2020) ^{S}	P100	100%	100%	1.25x	1.28x‡	83.9	89.2	93.4	–	−0.26
	(Goyal et al., 2020) ^{S}	K80	100%	100%	2.5x	3.1x‡	83.8	–	92.1	–	−1.1
Param. Sharing	(Lan et al., 2020) ^{A}	–	10.7%	8.8%	1.2x‡	1.2x‡	84.3	89.6	90.3	89.3	−0.58
Pruning with KD	(Mao et al., 2020) ^{S}	–	40.0%	37.3%	1x	1x	83.5	88.9	92.8	–	−0.7
	(Hou et al., 2020) ^{S}	K40	31.2%	12.4%	5.9x‡	8.7x‡	82.0	90.4	92.0	–	−0.96
Quantization with KD	(Zadeh et al., 2020) ^{S}	CPU	7.6%	3.9%	1.94x	1.73x	82.0	–	–	–	−2.6
	(Sun et al., 2020b)† ^{A}	Pixel	5.7%	6.1%	3.9x‡	4.7x‡	83.3	–	92.6	90.0	−0.23
Compound	(Tambe et al., 2020) ^{S}	TX2	1.3%	0.9%	1.83x	–	84.4	89.8	88.5	–	−1.53

Table 1: Evaluation of various compression methods. * indicates models using task-specific sizes or speedups; average values are reported in such cases. † represents models that use $\text{BERT}_{\text{LARGE}}$ as the teacher model. ‡ represents speedup values that we calculated. Empty cells in the speedup columns are for papers that do not describe the detailed architecture of their final compressed model. ^{A} marks models compressed in a task-agnostic setup, i.e., requiring access to the pre-training dataset. ^{S} indicates models compressed in a task-specific setup. *V100* is Nvidia Tesla V100; *P100* is Nvidia Tesla P100; *K80* is Nvidia Tesla K80; *Titan V* is Nvidia Titan V; *K40* is Nvidia Tesla K40; *CPU* is Intel Xeon E5; *TX2* is Nvidia Jetson TX2; and *Pixel* is Google Pixel Phone.

Quantization and Pruning. Quantization is well suited for BERT, and it can outperform other methods in terms of both model size and accuracy. As shown in Table 1, it can compress BERT to 15% and 10.2% of its original size, with accuracy drop of only 0.6% and 0.9%, respectively, across various tasks (Shen et al., 2020; Zadeh et al., 2020). This can be attributed to its architecture-invariant nature, as it only reduces the precision of the weights, but preserves all original components and connections. Unstructured pruning also shows performance that is on par with other methods. It compresses BERT to 67.6% of its original size, without any loss in accuracy, possibly due to the regularization effect of pruning (Guo et al., 2019). However, almost all existing work in unstructured pruning freezes the embedding matrix and focuses only on pruning the weight matrices of the encoder. This makes extreme compression difficult—for example, even with 3% weight density in encoders, the total model size still remains at 23.8% of its original size (Sanh et al., 2020), and yields a sizable drop in accuracy/F1 (4.73% on average).

While both quantization and unstructured pruning reduce the model size significantly, none of them yields actual run-time speedups on a standard device. Instead, specialized hardware and/or

libraries are required, which can do lower-bit arithmetic for quantization and an optimized implementation of sparse weight matrix multiplication for unstructured pruning. However, these methods can be easily combined with other compression methods as they are orthogonal from an implementation viewpoint. Below, we discuss the performance of compounding multiple compression methods.

Structured Pruning. As discussed in Section 3, structured pruning removes architectural components from BERT, which can also be seen as reducing the number of hyper-parameters that govern the BERT architecture. While Lin et al. (2020) pruned the encoder units (L) and reduced the model depth by half with an average accuracy drop of 1.0%, Khetan and Karnin (2020) took it a step further and systematically reduced both the depth (L) as well as the width (H, A) of the model, compressing to 39.1% of the original size with an average accuracy drop of only 1.86%. Detailed experiments by Khetan and Karnin (2020) also show that reducing all hyper-parameters in harmony, instead of focusing on just one, yields better performance.

Model-Agnostic Distillation. Applying distillation from output logits only allows model-agnostic compression and gives rise to LSTM/CNN-based student models. While methods exist that try to train a smaller BERT model (Song et al., 2020), this category is dominated by methods that replace Transformers with lighter alternatives. It has been shown that a BiLSTM student model can yield significantly better speedup (Liu et al., 2019a) compared to a Transformer-based student model of comparable size (Song et al., 2020). Chen et al. (2020a) demonstrated the fastest model in this category, a NAS-based CNN model, with only 2.06% average drop in accuracy. Overall, these methods achieved high compression ratio, but they paid a heavy price: sizable drop in accuracy. This could be because the total model size is not a true indicator of how powerful their compression is, as the model size is dominated by the embedding matrix.

For example, while the total size of the student model of Liu et al. (2019a) is 101 MB, only 11 MB is the size of their BiLSTM model, and the remaining 90 MB are just the embedding matrix. Thus, we can conclude that, similarly to unstructured pruning, ignoring the embedding matrix can hurt the practical deployment of such models on devices with strict memory constraints.

Distillation from Attention Maps. Wang et al. (2020c) were able to reduce BERT to 60.7% its original size, with only 0.1% loss in accuracy on average, just by doing deep distillation on the attention layers. For the same student architecture, Sanh et al. (2019) used all other forms of distillation (i.e., output logits and encoder outputs) together and still faced an average accuracy loss of 1.73%. Clearly, the intermediate attention maps are an important distillation target.

Combining Multiple Distillations. Combining multiple distillation targets can yield an even better compressed model. Jiao et al. (2020) created a student model with smaller H and L hyper-parameter values, compressing the model size to 13.3% and achieving a 9.4x speedup on a GPU (9.3x on a CPU), while only facing a drop of 1.0% in accuracy. Zhao et al. (2019b) extended the idea and created an extremely small BERT student model (1.6% of the original size, $\sim$ 25x faster) with $H = 48$ and vocabulary size $|V| = 4,928$ ($\text{BERT}_{\text{BASE}}$ has $H = 768$ and $|V| = 30,522$). The model lost 12.3% accuracy to pay for its size.

Matrix Decomposition and Dynamic Inference Acceleration. While weight matrix decomposition helps reduce the size of the weight matrices in BERT, it creates deeper and fragmented models, which hurts the execution time (Noach and Goldberg, 2020). On the other hand, methods that implement faster attention and various forms of dynamic speedup do not change the model size, but instead provide faster inference. For example, Cao et al. (2020) showed that attention calculation across the complete sentence is not needed for the initial encoder layers, and they were able to achieve $\sim$ 3x speedup with only 0.76% drop in accuracy. For applications where latency is the major constraint, such methods can be suitable.

Structured Pruning vs. Distillation. While structured pruning attempts to iteratively prune the hyper-parameters of BERT, distillation starts with a smaller model and tries to train it using knowledge directly from the original BERT. However, both of them end up with a similar compressed model, and thus it is interesting to compare which path yields better results. As can

be noted from Table 1, for the same compressed model with $L = 6$, the drop in accuracy for the model of Lin et al. (2020) is smaller compared to that of Sanh et al. (2019). However, this is not a completely fair comparison, as Sanh et al. (2019) did not use attention as a distillation target. When we compare other methods, we find that Jiao et al. (2020) was able to beat Khetan and Karnin (2020) in terms of both model size and accuracy. This shows that structured pruning outperforms student models trained using distillation only on encoder outputs and output logits, but fails against distillation on attention maps. This further indicates the importance of replicating attention maps in BERT.

Pruning with Distillation. Similarly to combining multiple distillation methods, it is also possible to combine pruning with distillation, as this can help guide the pruning towards removing the less important connections. Mao et al. (2020) combined distillation with unstructured pruning, while Hou et al. (2020) combined distillation with structured pruning. When compared with only structured pruning (Khetan and Karnin, 2020), we see that Hou et al. (2020) achieved both a smaller model size (12.4%) and also a smaller drop in accuracy (0.96%).

Quantization with Distillation. Similarly to pruning, quantization is also orthogonal in implementation to distillation, and can together achieve better performance than either of them individually. Zadeh et al. (2020) attempted to quantize an already distilled BERT model (Sanh et al., 2019) to four bits, thus reducing the model size from 60.2% to 7.5%, with an additional accuracy drop of only 0.9% (1.73% to 2.6%). Similarly, Sun et al. (2020b) attempted to quantize their model to eight bits, which reduced their model size from 23% to 5.25%, with only a 0.07% additional drop in accuracy.

Compounding Multiple Methods Together. As we have seen in this section, different methods of compression target different parts of the BERT architecture. Note that many of these methods are orthogonal in implementation, similarly to the work we discussed on combining quantization and pruning with distillation, and thus it is possible to combine them. For example, Tambe et al. (2020) combined multiple forms of compression methods to create a truly deployable language model for edge devices. They combined parameter sharing, embedding matrix decomposition, unstructured movement pruning, adaptive floating-point quantization, adaptive attention span, dynamic inference speed with early exit ramps, and other hardware accelerations to suit their needs. However, as we noticed in this section, these particular methods can reduce the model size significantly, but they cannot drastically speed up the model execution on standard devices. While the model size is reduced to only 1.3% of its original size, the speedup obtained on a standard GPU is only 1.83x, with an average drop of 1.53% in terms of accuracy. With specialized accelerators, the authors eventually pushed the speedup to 2.1x.

4.3 Practical Advice

Based on the experimental results we have discussed in this section, below we attempt to give some practical advice to the reader on what to use for specific applications:

- Quantization and unstructured pruning can help reduce the model size, but they do nothing to improve the runtime inference speed or the memory consumption, unless executed on specialized hardware or with specialized processing libraries. On the other hand, if executed on proper hardware, these methods can provide tremendous boost in terms of speed with negligible loss in performance (Zadeh et al., 2020; Tambe et al., 2020; Guo et al., 2019). Thus, it is important to recognize the target hardware device before deciding to use such compression methods in practical applications.

- Knowledge distillation has shown great affinity to a variety of student models and its orthogonal nature of implementation compared to other methods (Mao et al., 2020; Hou et al., 2020) means that it is an important addition to any form of compression. More specifically, distillation from self-attention layers (if possible) is an integral part of Transformer compression (Wang et al., 2020c).

- Alternatives such as BiLSTMs and CNNs have an additional advantage in terms of execution speed when compared to Transformers. Thus, for applications with strict latency constraints, replacing Transformers

with alternative units is a better choice. Model execution can also be sped up using dynamic inference methods, as they can be incorporated into any student model with a skeleton that is similar to that of Transformers.

- A major takeaway of our discussion above is the importance of compounding various compression methods together to achieve truly practical models for edge environments. The work of Tambe et al. (2020) is a good example of this, as it attempts to compress BERT, while simultaneously performing hardware optimizations in accordance with their chosen compression methods. Thus, combining compression methods that complement each other is generally a better idea than compressing a single aspect of the model to its extreme.

5 Open Issues and Research Directions

From our analysis and comparison, we conclude that traditional model compression methods such as quantization and pruning are beneficial for BERT. Techniques specific to BERT also yield competitive results, for example, variants of knowledge distillation and methods that reduce the number of architectural hyper-parameters. Such methods also offer insights into BERT's workings and the importance of various layers in its architecture. We see multiple avenues for future research:

1. A very prominent feature of most BERT compression methods is their coupled nature across various encoder units, as well as the inner architecture. However, some layers might be able to handle more compression. Methods compressing each layer independently (Khetan and Karnin, 2020; Tsai et al., 2020) have shown promising results, but remain under-explored.

2. The Transformer backbone that forces the model to be parameter-heavy makes compression challenging. Existing work in replacing the Transformer by Bi-LSTMs and CNNs has yielded extraordinary compression ratios, but with a sizable drop in accuracy. This suggests further exploration of more complex variations and hybrid Bi-LSTM/CNN/Transformer models (Tian et al., 2019).

3. Many methods for BERT compression only work on specific parts of the model. However, we can combine such methods to achieve better results. We have seen in Section 4 that compound compression methods perform better than their individual counterparts (Tambe et al., 2020; Hou et al., 2020), and thus more exploration in combining various existing methods is needed.

Acknowledgments

This publication was made possible by NPRP grant NPRP10-0208-170408 from the Qatar National Research Fund (a member of Qatar Foundation). This work is also partially supported by the National Research Foundation, Prime Minister's Office, Singapore, under its Campus for Research Excellence and Technological Enterprise (CREATE) program. The findings herein reflect the work, and are solely the responsibility of, the authors.

References

Yoonho Boo and Wonyong Sung. 2020. Fixed-point optimization of transformer neural network. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, ICASSP '20, pages 1753–1757.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33 of *NeurIPS '20*, pages 1877–1901.

Qingqing Cao, Harsh Trivedi, Aruna Balasubramanian, and Niranjan Balasubramanian. 2020. DeFormer: Decomposing pre-trained transformers for faster question answering. In *Proceedings of the 58th Annual Meeting of*

the Association for Computational Linguistics, ACL '20, pages 4487–4497.

Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. 2020a. AdaBERT: Task-adaptive BERT compression with differentiable neural architecture search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI* 2020, pages 2463–2469. https://doi.org/10.24963/ijcai.2020/341

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michaelg Carbin. 2020b. The lottery ticket hypothesis for pre-trained BERT networks. In *Proceedings of the 34th Conference on Neural Information Processing Systems*, NeurIPS '20, pages 1753–1757, Vancouver, Canada.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2018. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136. https://doi.org/10.1109/MSP.2017.2765695

Yew Ken Chia, Sam Witteveen, and Martin Andrews. 2018. Transformer to CNN: Label-scarce distillation for efficient text classification. In *Proceedings of the Compact Deep Neural Network Representation with Industrial Applications Workshop*, Montréal, Canada.

Artem Chumachenko, Daniil Gavrilov, Nikita Balagansky, and Pavel Kalaidin. 2020. Weight squeezing: Reparameterization for extreme compression and fast inference. *arXiv:2010.06993*.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? An analysis of BERT's attention. In *Proceedings of the ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, BlackboxNLP'19, pages 276–286. Florence, Italy. https://doi.org/10.18653/v1/W19-4828

Kalyanmoy Deb. 2014. Multi-objective optimization. In *Search Methodologies*, pages 403–449. Springer. https://doi.org/10.1007/978-1-4614-6940-7_15

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT '19, pages 4171–4186, Minneapolis, MN, USA.

Lifang Ding and Yujiu Yang. 2020. SDSK2BERT: Explore the specific depth with specific knowledge to compress BERT. In *Proceedings of the IEEE International Conference on Knowledge Graph*, ICKG '20, pages 420–425. https://doi.org/10.1109/ICBK50248.2020.00066

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *Proceedings of the 8th International Conference on Learning Representations*, ICLR '20, Addis Ababa, Ethiopia.

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Hervé Jégou, and Armand Joulin. 2021. Training with quantization noise for extreme model compression. In *Proceedings of the 10th International Conference on Learning Representations*, ICLR '21.

Luciano Floridi and Massimo Chiriatti. 2020. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30(4):681–694. https://doi.org/10.1007/s11023-020-09548-1

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the 7th International Conference on Learning Representations*, ICLR '19, New Orleans, LA, USA.

Mitchell Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing BERT: Studying the effects of weight pruning on transfer learning. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, RepL4NLP '20, pages 143–155. https://doi.org/10.18653/v1/2020.repl4nlp-1.18

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy,

Yogish Sabharwal, and Ashish Verma. 2020. PoWER-BERT: Accelerating BERT inference via progressive word-vector elimination. In *Proceedings of the International Conference on Machine Learning*, ICML '20, pages 3690–3699.

Fu-Ming Guo, Sijia Liu, Finlay S. Mungall, Xue Lin, and Yanzhi Wang. 2019. Reweighted proximal pruning for large-scale language representation. *arXiv:1909.12486*.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. DynaBERT: Dynamic BERT with adaptive width and depth. In *Advances in Neural Information Processing Systems*, volume 33 of *NeuIPS '20*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, ACL '18, pages 328–339, Melbourne, Australia. https://doi.org/10.18653/v1/P18-1031

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, EMNLP '20, pages 4163–4174. https://doi.org/10.18653/v1/2020.findings-emnlp.372

Ashish Khetan and Zohar Karnin. 2020. schuBERT: Optimizing elements of BERT. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, ACL'20, pages 2807–2818. https://doi/10.18653/v1/2020.acl-main.250

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *Proceedings of the International Conference on Learning Representations*, ICLR '20, Addis Ababa, Ethiopia.

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. 2019. Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, EMNLP-IJCNLP '19, pages 4365–4374, Hong Kong, China.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *Proceedings of the 8th International Conference on Learning Representations*, ICLR '20, Addis Ababa, Ethiopia.

Bingbing Li, Zhenglun Kong, Tianyun Zhang, Ji Li, Zhengang Li, Hang Liu, and Caiwen Ding. 2020a. Efficient transformer-based large scale language representations using hardware-friendly block structured pruning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3187–3199.

Jianquan Li, Xiaokang Liu, Honghong Zhao, Ruifeng Xu, Min Yang, and Yaohong Jin. 2020b. BERT-EMD: Many-to-many layer mapping for BERT compression with earth mover's distance. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, EMNLP '20, pages 3009–3018.

Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. 2020c. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *Proceedings of the International Conference on Machine Learning*, ICML '20, pages 5958–5968.

Zi Lin, Jeremiah Liu, Zi Yang, Nan Hua, and Dan Roth. 2020. Pruning redundant mappings in transformer models via spectral-normalized identity prior. In *Findings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 719–730. https://doi.org/10.18653/v1/2020.findings-emnlp.64

Linqing Liu, Huan Wang, Jimmy Lin, Richard Socher, and Caiming Xiong. 2019a. MKD: A

multi-task knowledge distillation approach for pretrained language models. *arXiv:1911.03588*.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. FastBERT: A self-distilling BERT with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, ACL '20, pages 6035–6044.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv:1907.11692*.

Matous Machacek and Ondrej Bojar. 2014. Results of the WMT14 metrics shared task. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, WMT '14, pages 293–301, Baltimore, MD, USA. https://doi.org/10.3115/v1/W14-3336

Yihuan Mao, Yujing Wang, Chufan Wu, Chen Zhang, Yang Wang, Quanlu Zhang, Yaming Yang, Yunhai Tong, and Jing Bai. 2020. LadaBERT: Lightweight adaptation of BERT through hybrid model compression. In *Proceedings of the 28th International Conference on Computational Linguistics*, COLING '20, pages 3225–3234.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, volume 32 of *NeurIPS '19*, pages 14014–14024, Vancouver, BC, Canada.

Subhabrata Mukherjee and Ahmed H. Awadallah. 2020. XtremeDistil: Multi-stage distillation for massive multilingual models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, ACL '20, pages 2221–2234. https://doi.org/10.18653/v1/2020.acl-main.202

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, EMNLP '18, pages 1797–1807, Brussels, Belgium. https://doi.org/10.18653/v1/D18-1206

Matan Ben Noach and Yoav Goldberg. 2020. Compressing pre-trained language models by matrix decomposition. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, AACL-IJCNLP '20, pages 884–889.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT '18, pages 2227–2237, New Orleans, LA, USA. https://doi.org/10.18653/v1/N18-1202

Prafull Prakash, Saurabh Kumar Shashidhar, Wenlong Zhao, Subendhu Rongali, Haidar Khan, and Michael Kayser. 2020. Compressing transformer-based semantic parsing models using compositional code embeddings. In *Findings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 4711–4717. https://doi.org/10.18653/v1/2020.findings-emnlp.423

Sai Prasanna, Anna Rogers, and Anna Rumshisky. 2020. When BERT plays the lottery, all tickets are winning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, EMNLP '20, pages 3208–3229. https://doi.org/10.18653/v1/2020.emnlp-main.259

XiPeng Qiu, TianXiang Sun, YiGe Xu, YunFan Shao, Ning Dai, and XuanJing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897. https://doi.org/10.1007/s11431-020-1647-3

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael

Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67.

Alessandro Raganato, Yves Scherrer, and Jörg Tiedemann. 2020. Fixed encoder self-attention patterns in transformer-based machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 556–568. https://doi.org/10.18653/v1/2020.findings-emnlp.49

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, ACL '18, pages 784–789, Melbourne, Australia. https://doi.org/10.18653/v1/P18-2124

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, EMNLP '16, pages 2383–2392, Austin, TX, USA. https://doi.org/10.18653/v1/D16-1264

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866. https://doi.org/10.1162/tacl_a_00349

Corby Rosset. 2020. Turing-NLG: A 17-billion-parameter language model by Microsoft. *Microsoft Research Blog*, 2:13.

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

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *Proceedings of the 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing*, Vancouver, Canada. https://doi.org/10.1609/aaai.v34i05.6409

Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems*, volume 33 of *NeurIPS '20*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: Hessian based ultra low precision quantization of BERT. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34 of *AAAI '20*, pages 8815–8821.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv:1909.08053*.

Kaitao Song, Hao Sun, Xu Tan, Tao Qin, Jianfeng Lu, Hongzhi Liu, and Tie-Yan Liu. 2020. LightPAFF: A two-stage distillation framework for pre-training and fine-tuning. *arXiv:2004.12817*.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, EMNLP-IJCNLP '19, pages 4314–4323, Hong Kong, China. https://doi.org/10.18653/v1/D19-1441

Siqi Sun, Zhe Gan, Yuwei Fang, Yu Cheng, Shuohang Wang, and Jingjing Liu. 2020a. Contrastive distillation on intermediate representations for language model compression. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, EMNLP '20, pages 498–508, https://doi.org/10.18653/v1/2020.emnlp-main.36

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020b. MobileBERT: A compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, ACL '20, pages 2158–2170.

Thierry Tambe, Coleman Hooper, Lillian Pentecost, Tianyu Jia, En-Yu Yang, Marco Donato, Victor Sanh, Paul Whatmough,

Alexander M Rush, David Brooks, and Gu-Yeon Wei. 2020. EdgeBERT: Sentence-level energy optimizations for latency-aware multi-task NLP inference. *arXiv:2011.14203*.

Raphael Tang, Yao Lu, and Jimmy Lin. 2019. Natural language generation for effective knowledge distillation. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP*, DeepLo '19, pages 202–208, Hong Kong, China. https://doi.org/10.18653/v1/D19-6122

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2020. Synthesizer: Rethinking self-attention in transformer models. *arXiv:2005.00743*.

James Yi Tian, Alexander P. Kreuzer, Pai-Hung Chen, and Hans-Martin Will. 2019. WaLDORf: Wasteless language-model distillation on reading-comprehension. *arXiv:1912.06638*.

Henry Tsai, Jayden Ooi, Chun-Sung Ferng, Hyung Won Chung, and Jason Riesa. 2020. Finding fast transformers: One-shot neural architecture search by component composition. *arXiv:2008.06808*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS '17, pages 6000–6010, Long Beach, CA, USA.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, BlackboxNLP '18, pages 353–355, Brussels, Belgium. https://doi.org/10.18653/v1/W18-5446

Shirui Wang, Wenan Zhou, and Chao Jiang. 2020a. A survey of word embeddings based on deep learning. *Computing*, 102(3):717–740.

Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. 2020b. Linformer: Self-attention with linear complexity. *arXiv:2006.04768*. https://doi.org/10.1007/s00607-019-00768-7

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020c. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, volume 33 of *NeurIPS '20*.

Moshe Wasserblat, Oren Pereg, and Peter Izsak. 2020. Exploring the boundaries of low-resource BERT distillation. In *Proceedings of the Workshop on Simple and Efficient Natural Language Processing*, SustaiNLP '20, pages 35–40. https://doi.org/10.18653/v1/2020.sustainlp-1.5

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Deanand. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144*.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, ACL '20, pages 2246–2251. https://doi.org/10.18653/v1/2020.acl-main.204

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. BERT-of-Theseus: Compressing BERT by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, EMNLP '20, pages 7859–7869.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R. Salakhutdinov, and Quoc V. Le. 2019. XLNet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32 of *NeuIPS '19*, pages 5753–5763.

Ali H. Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. 2020. GOBO:

Quantizing attention-based NLP models for low latency and energy efficient inference. In *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '20, pages 811–824. https://doi.org/10.1109/MICRO50266.2020.00071

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8bit BERT. In *Proceedings of the 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing*, Vancouver, Canada. https://doi.org/10.1109/EMC2-NIPS53020.2019.00016

Guangxiang Zhao, Junyang Lin, Zhiyuan Zhang, Xuancheng Ren, Qi Su, and Xu Sun. 2019a. Explicit sparse transformer: Concentrated attention through explicit selection. *arXiv: 1912.11637*.

Sanqiang Zhao, Raghav Gupta, Yang Song, and Denny Zhou. 2019b. Extreme language model compression with optimal subwords and shared projections. *arXiv:1909.11687*.

Denny Zhou, Mao Ye, Chen Chen, Tianjian Meng, Mingxing Tan, Xiaodan Song, Quoc Le, Qiang Liu, and Dale Schuurmans. 2020a. Go wide, then narrow: Efficient training of deep thin networks. In *Proceedings of the International Conference on Machine Learning*, ICML '20, pages 11546–11555.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020b. BERT loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*, volume 33 of *NeurIPS '20*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision*, ICCV '19, pages 19–27, Seoul, Korea. https://doi.org/10.1109/ICCV.2015.11