A Survey on Model Compression and Acceleration for Pretrained Language Models

Canwen Xu, Julian McAuley

University of California, San Diego
{cxu, jmcauley}@ucsd.edu

Abstract

Despite achieving state-of-the-art performance on many NLP tasks, the high energy cost and long inference delay prevent Transformer-based pretrained language models (PLMs) from seeing broader adoption including for edge and mobile computing. Efficient NLP research aims to comprehensively consider computation, time and carbon emission for the entire life-cycle of NLP, including data preparation, model training and inference. In this survey, we focus on the inference stage and review the current state of model compression and acceleration for pretrained language models, including benchmarks, metrics and methodology.

Introduction

The recent success of applying pretrained deep Transformers (Vaswani et al. 2017) on different NLP tasks (Devlin et al. 2019; Raffel et al. 2020; Le Scao et al. 2022) has raised concerns about its efficiency. The high computational cost also prevents these pretrained language models (PLMs) from being deployed in production (Sun et al. 2020). To address this problem, *efficient inference* refers to techniques that aim to make inference of an ML model faster (time-efficient), consume fewer computational resources (computation-efficient), less memory (memory-efficient) and less disk space (storage-efficient). One popular class of techniques is *model compression and acceleration*, where a large and slow model is compressed to a lightweight model that can be stored with limited disk space on a mobile device, or accelerated to run with low latency (or both). Also, training a large model and then compressing it to a small one can be efficient for training and good for generalization (Li et al. 2020).

In addition to technical considerations, large models also raise environmental and ethical concerns (Bender et al. 2021). Large models have a high carbon footprint which a compressed model can reduce, potentially with little sacrifice in performance. Meanwhile, large models set obstacles for engineers and researchers from developing countries who cannot afford the necessary hardware for running the model (Bender et al. 2021). Thus, model compression and acceleration can be critical to make state-of-the-art NLP techniques more accessible and facilitate inclusiveness.

What's covered? In this survey, we aim to highlight the most important works in the field of model compression and acceleration for PLMs. We review the metrics, benchmarks, and methods, organizing these works in a new taxonomy. Widely-used techniques, including weight sharing, low-rank factorization, pruning, quantization, knowledge distillation, early exit, and token skipping, are covered with comparative analysis. We also highlight current challenges and future research directions in the field, calling for community efforts to build an environmentally-friendly, inclusive and sustainable future of NLP.

What's not covered? This survey does not cover (1) methods that design a new architecture for training from scratch (e.g., long-range Transformers, Mixture-of-Experts models); (2) data-efficient or parameter-efficient model tuning that focuses more on the training efficiency rather than inference efficiency (e.g., few-shot learning, prompt learning, partial model tuning); (3) works that use the techniques surveyed in this paper but for other purposes or are application-specific (e.g., self-distillation, representation distillation for retrieval).

There have been surveys (Qiu et al. 2020; Han et al. 2021; Xu et al. 2021b) that cover some aspects of this topic. Different from these works, we focus on the latest progress on model compression and acceleration for pretrained language models, highlighting the intersection between language technology and efficient ML.

Metrics and Benchmarks

Metrics

There are various metrics to depict inference efficiency in different dimensions. These metrics are often reported together with accuracy to evaluate an NLP model.

Floating point operations (FLOPs) directly measures the number of floating points operations needed for executing an instance. FLOPs can serve as a metric for computational efficiency and is somewhat hardware-agnostic. However, FLOPs cannot accurately reflect the real runtime since the degree of parallelism (DOP) varies for different algorithms.

Inference time (i.e., delay) is used to measure the runtime of an algorithm in its inference stage. Inference time can vary on different infrastructures. When testing on the same architecture, compared to FLOPs, inference time can better approximate the real-world performance of a system by

taking parallelism into consideration.

Speed-up Ratio is the ratio of inference time of the baseline model to the accelerated model. Compared to inference time, speed-up ratio draws a relative comparison which can be roughly compared across different hardware. In some works (Zhou et al. 2020; Sun et al. 2021), speed-up ratio is approximated by the ratio of the number of Transformer layers in the baseline model to those used in calculation of an acceleration method.

Number of Parameters and Model Size are often reported in NLP studies as metrics that directly reflect the storage cost of a model. This can be important for mobile deployment of an NLP model due to limited storage on a mobile device. It can also be an indicator for the memory footprint and computational cost for training and inference. An exception is models with weight sharing. For example, the FLOPs and memory use of ALBERT (Lan et al. 2020) is slightly higher than a BERT model (Devlin et al. 2019) with the same number of layers. However, since all Transformer layers in ALBERT share the same weights, the model size of n-layer ALBERT is only $1/n$ of n-layer BERT.

Carbon Footprint measures the environmental impact. Lacoste et al. (2019) provide a calculator for CO_2 by querying a database of carbon emission of mainstream cloud computing providers. Alternatively, Experiment Impact Tracker (Henderson et al. 2020) and CodeCarbon[1] are two plugins that can record the energy use of hardware and estimate the carbon emission based on the geolocation.

Loyalty/Fidelity Recent works Xu et al. (2021a) and Stanton et al. (2021) propose *loyalty* and *fidelity*, respectively. Both are similarity metrics calculated between the predicted distributions of the teacher and the student, in a teacher-student distillation or compression setting. *Loyalty* and *fidelity* can reflect how successful the knowledge transfer is from the teacher to the student, providing interpretability and reliability (Xu et al. 2021a; Stanton et al. 2021), and can be an indicator of better generalization in distilling large teacher models and ensembles (Stanton et al. 2021).

Robustness Su et al. (2018) find that smaller neural networks are more vulnerable to adversarial attacks. Xu et al. (2021a) suggest reporting adversarial robustness in addition to accuracy. In addition to adversarial robustness, Du et al. (2021) find compressed pretrained language models are significantly less robust on out-of-distribution (OOD) data.

Benchmarks

Standard Benchmarks Most studies evaluate on common NLP benchmarks. For example, GLUE (Wang et al. 2019b) and SuperGLUE (Wang et al. 2019a) are used for natural language understanding (NLU). SQuAD (Rajpurkar et al. 2016) is used for machine reading comprehension (MRC).

EfficientQA EfficientQA (Min et al. 2020) is an open-domain question answering benchmark encouraging solutions that efficiently store and access knowledge with the smallest number of bytes. EfficientQA has three resource-restricted tracks, including two tracks with a 6 GB and 500

---

[1]https://codecarbon.io/

MB cut-off for system size and one track that ranks the systems that achieves 25% accuracy with the smallest size.

SustaiNLP The shared task of SustaiNLP 2020 (Wang and Wolf 2020) uses SuperGLUE (Wang et al. 2019a) to evaluate the performance of submissions. There are three tracks that target different accuracy levels and hardware (2 GPU tracks and 1 CPU track). Within each track, submissions are ranked by lowest energy consumption, measured by Experiment Impact Tracker (Henderson et al. 2020).

ELUE Efficient Language Understanding Evaluation (Liu et al. 2021) is proposed as an attempt to clearly depict the Pareto Front of FLOPs versus performance. ELUE consists of six datasets of three tasks (sentiment analysis, natural language inference, and paraphrasing). ELUE has four tracks with a parameter number cut-off of 40M, 55M, 70M and 110M. The metric used for evaluation is ELEU score, which calculates an average performance advantage over a baseline (ElasticBERT) under different FLOPs.

Methods

Weight Sharing

Weight sharing is based on the assumption that large-scale models, like Transformer (Vaswani et al. 2017), are over-parameterized (Li et al. 2020). Weight sharing provides a way to decouple computation and parameters by reusing the same parameters for multiple computations. Weight sharing can reduce inference memory footprint and number of parameters and thus is memory- and storage-efficient.

Encoder-Decoder Sharing In the vanilla Transformer model (Vaswani et al. 2017) for neural machine translation (NMT), there is one encoder for encoding the input into hidden representations, and one decoder for decoding it to the target language. Tied Transformer (Xia et al. 2019) shares the weights of the encoder and decoder of Transformer. The results of Tied Transformer are comparable to the vanilla Transformer. Rothe, Narayan, and Severyn (2020) leverage pretrained language model checkpoints to initialize a sequence-to-sequence model. They experiment with a shared encoder and decoder to reduce memory footprint.

Layer Sharing In Transformer (Vaswani et al. 2017), both the encoder and decoder are stacks of Transformer layers. Thus, a simple and straightforward way to share the weights in a Transformer is to share them across all Transformer layers. Dabre and Fujita (2019) share the weights across all Transformer layers for NMT with minimal performance drop. Universal Transformer (Dehghani et al. 2019) shares the weights across all layers, allowing for recurrent computation with a dynamic halting mechanism and achieves better performance than the vanilla Transformer. ALBERT (Lan et al. 2020) introduces the idea into pretrained language models for natural language understanding (NLU). Although it cannot reduce the computational overheads and has an inevitable negative effect on performance, this design saves up to 95% of disk space for storing the model, which can be critical for deployment on mobile devices with limited storage. Takase and Kiyono (2021) systematically study strategies for sharing weights across layers. Instead of using the weights of one Transformer layer for all layers, they aim to explore the

	Hessian-based pruning (1989)	Magnitude pruning (2016)	L_0 regularization (2018)	Movement pruning (2020)
Decision	2nd order	0th order	1st order	1st order
Objective	$\mathcal{L}$	$\mathcal{L}$	$\mathcal{L} + \lambda_{l0}\mathbb{E}(L_0)$	$\mathcal{L}$
Scores **S**	$-\sum_t (\frac{\partial^2 \mathcal{L}}{\partial W^2_{i,j}})^{(t)} W_{i,j}^{2(t)}$	$\lvert W_{i,j}\rvert$	$-\sum_t (\frac{\partial \mathcal{L}}{\partial W_{i,j}})^{(t)} W_{i,j}^{(t)} f(\overline{S}_{i,j}^{(t)})$	$-\sum_t (\frac{\partial \mathcal{L}}{\partial W_{i,j}})^{(t)} W_{i,j}^{(t)}$

Table 1: A summary of various pruning methods. **S** are saliency scores used to determine which weights to prune. The table style is borrowed from Sanh, Wolf, and Rush (2020).

best way to use the parameters of M layers for N layers ($M < N$). Reid, Marrese-Taylor, and Matsuo (2021) introduce a strategy named "sandwich-style" parameter sharing, which shares the weights for central layers while leaving the first and last layers independent.

Low-Rank Factorization

The weight matrices in a neural network are often low-rank, indicating redundancy in model weights (Sainath et al. 2013). Thus, a natural idea is to factorize the weight matrices into two or more smaller matrices to save parameters. A common technique for low-rank factorization is singular value decomposition (SVD). For a matrix $A \in \mathbb{R}^{m \times n}$, there exists $A = U\Sigma V^{\mathrm{T}}$, where $r \leq \min\{m, n\}$ is the rank of A; $U \in \mathbb{R}^{m \times r}$, $V \in \mathbb{R}^{n \times r}$ are two orthogonal matrices; $\Sigma \in \mathbb{R}^{r \times r}$ is a diagonal matrix with only the non-zero singular values of A. Thus, the space complexity can be effectively reduced from $O(mn)$ to $O(mr + rn)$, improving the storage-efficiency of the model.

Decomposing Linear Layers Low-rank factorization can be applied to any linear layer. Grachev, Ignatov, and Savchenko (2017) factorize the weights of an LSTM language model. Following that, Winata et al. (2019) exploit SVD for both the LSTM cell in a language modeling task and a pretrained LSTM language model, ELMo (Peters et al. 2018). This is one of the earliest attempts to compress a pretrained language model. Ma et al. (2019) propose a new self-attention module, namely multi-linear attention, as a substitute for the standard multi-head attention module in a Transformer. They use block-term tensor decomposition (BTD, Lathauwer 2008) to factorize multi-head attention. Their results demonstrate comparable performance to the vanilla Transformer while being parameter-efficient. Noach and Goldberg (2020) propose a two-stage approach to compress a pretrained language model. In the first stage, they decompose each weight matrix in the pretrained language model with SVD. Then, for the second stage, they fine-tune or use knowledge distillation to refine the weights. Chen et al. (2021) propose data-aware low-rank compression (DRONE) by exploiting the prior knowledge of the data distribution. Instead of minimizing the reconstruction error of the the weight matrix, they minimize the approximation error of the outputs. DRONE achieves better performance than SVD. Besides, as an alternative to SVD, Kronecker decomposition retains the rank of the matrix and has shown improvement compressing BERT and GPT-2 (Tahaei et al. 2021; Edalati et al. 2022).

Decomposing Embedding ALBERT (Lan et al. 2020) uses factorization for the embedding layer, which has redundant parameters due to its high input and output dimensions. Since the power of Transformer mainly comes from its contextual learning ability, the parameters in the token embedding layer are not efficient. It intuitively makes sense to reduce them by factorizing the embedding matrix. Reid, Marrese-Taylor, and Matsuo (2021) propose self-attentive factorized embeddings (SAFE) by adding a small self-attention layer on the basis of linear projection to achieve better performance.

Pruning

Pruning (LeCun, Denker, and Solla 1989) aims to remove unimportant weights from a neural network to achieve storage-, memory-efficiency, and sometimes also computation- and time-efficiency while preserving model performance. There are two key elements in a pruning method: (1) A **pruning unit** is the atomic unit to be removed from a model; it can be a single weight (unstructured pruning), an attention head or even a Transformer layer (structured pruning). (2) A **saliency score** is the criterion for making pruning decisions. Based on whether it uses a gradient and which order of gradient it uses, we can categorize pruning methods to zeroth-order (only considering weight magnitude), first-order and second-order approaches. We summarize some representative pruning methods in Table 1.

Unstructured Pruning Unstructured pruning removes "unimportant" connections in a neural network by setting the corresponding weights to 0. After pruning, the weight matrix often becomes sparse. To exploit the characteristics of a sparse matrix to achieve computation- and memory-efficiency, specialized hardware (e.g., sparse tensor cores in Nvidia A100 (Mishra et al. 2021)) and software (e.g., PyTorch Sparse API[2]) are necessary. See, Luong, and Manning (2016) uses magnitude-based pruning with retraining to compress RNN models for NMT. Magnitude-based pruning (Han, Mao, and Dally 2016) simply prunes weights with smallest magnitude (i.e., absolute values). After pruning, See, Luong, and Manning (2016) continue to fine-tune the pruned network to obtain better performance. Narang et al. (2017) prune an RNN model gradually during training. The magnitude threshold for pruning is gradually increased with increasing training steps. Wang et al. (2020b) first prunes and retrains NMT models with magnitude pruning and then restores the pruned parameters to train the entire network again, in order to achieve better performance than the original model. Zhang and Stadie (2020) propose a one-shot pruning technique based on the

---
[2]https://pytorch.org/docs/stable/sparse.html

Jacobian spectrum. Different from iterative pruning methods, one-shot pruning techniques only prune a network once and then use standard training to train the sparse network.

Some recent works target transfer learning as it has become the new standard paradigm in NLP. Gordon, Duh, and Andrews (2020) aim to reveal how pruning affects transfer learning. They find that low levels of pruning (30%–40%) do not affect pretraining loss or transfer to downstream tasks at all. However, further pruning has a negative impact on both pretraining and transfer learning. A high level of pruning can even prevent the model from fitting downstream datasets. Sanh, Wolf, and Rush (2020) claim that magnitude pruning is suboptimal for transfer learning. They propose *movement pruning* as a simple first-order method for fine-tuning of pretrained language models. Instead of preserving weights that are currently far from zero, they retain those that are moving away from zero (i.e., gaining larger magnitude) during fine-tuning, achieving better performance for pruning BERT. Guo, Rush, and Kim (2021) propose *diff pruning*, by learning a task-specific "diff" vector that extends the original pretrained parameters. The task-specific "diff" vectors are trained with L_0 regularization (Louizos, Welling, and Kingma 2018) to encourage sparsity. By updating only 0.5% of parameters, diff pruning achieves similar performance to fine-tuning the whole network.

Structured Pruning Structured pruning removes weight blocks, rows, attention heads, or layers from a model. Compared to unstructured pruning, it can usually achieve acceleration and memory reduction without specialized hardware or software. Narang, Undersander, and Diamos (2017) extends gradual pruning for RNNs (Narang et al. 2017) to structured pruning. They first divide weights into blocks, then prune blocks of weights in a layer using group lasso regularization (Yuan and Lin 2006) to create blocks of zeros in weight matrices. Michel, Levy, and Neubig (2019) and Voita et al. (2019) find that the multi-head attention in Transformer has redundancy. Both works use a first-order approach to remove attention heads from Transformer. Following this, Fan, Grave, and Joulin (2020) propose a structured dropout strategy named LayerDrop. When training a Transformer, a random dropout is applied to each layer. After one-off training, the model can be pruned on-demand to achieve the target inference speed. SNIP (Lin et al. 2020) removes unimportant non-linear terms in the residual connections, whose magnitude is below a threshold. Lagunas et al. (2021) introduce a block pruning approach that extends structured methods by considering blocks of any size. They integrate this structure into movement pruning (Sanh, Wolf, and Rush 2020) and find this approach can automatically learn to prune out full components in Transformer, e.g., an attention head. Xia, Zhong, and Chen (2022) propose CoFi, a pruning method that jointly prunes both coarse-grained units (e.g., layers) and fine-grained units (e.g., attention head and hidden units). CoFi also introduces a distillation loss to further improve its performance.

Lottery Ticket Hypothesis Frankle and Carbin (2019) propose the "lottery ticket hypothesis": dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that — when trained in isolation — reach test accuracy comparable to the original network in a similar number of iterations. It reveals that retraining from remaining weights (Han, Mao, and Dally 2016) in a pruned network is not necessary for the pruned network. In contrary, a "winning ticket" can always learn better, even when training from scratch (as long as it is initialized with the same random weights). Following this, Chen et al. (2020) verify the lottery ticket hypothesis on BERT with iterative magnitude pruning. They find that subnetworks found on the pretraining task (i.e., masked language modeling, MLM) transfer universally to downstream tasks whereas those found on downstream tasks do not. Prasanna, Rogers, and Rumshisky (2020) also verify the lottery ticket hypothesis with BERT, for both magnitude and structured pruning. They find that even the worst subnetwork in BERT remains highly trainable, suggesting that the weights of BERT may have relatively low redundancy. This seems to be consistent with previous finding on over-parameterized models (Nakkiran et al. 2020).

Quantization

Quantization aims to compress a neural network by reducing the number of bits (i.e., precision) in the weights of the model, improving storage-, memory-, computation-, and time-efficiency (on most hardware). In general, quantization can be further categorized into post-training quantization (PTQ) and quantization-aware training (QAT). PTQ rescales the weights of a trained model whereas QAT introduces the rounding error into the training process. Due to the considerable performance drop for PTQ, most works in compressing NLP models unanimously use QAT, in order to achieve comparable performance with the full-precision model.

8-Bit Quantization Quantizing models from full precision floats (FP32) to 8-bit integers (INT8) is a classical setting, since operations including matrix multiplication can be calculated much faster with INT8 than FP32, especially on a CPU. Zafrir et al. (2019) use symmetric linear quantization (Jacob et al. 2018) to quantize both weights and activations to INT8 dynamically. They also explore quantization-aware training (QAT) for BERT. They use *fake quantization* (Jacob et al. 2018) to introduce quantization error into the model during training phase to simulate the rounding effect. They use Straight-Through Estimator (STE) (Bengio, Léonard, and Courville 2013) to estimate the gradient of the non-differentiable fake quantization. They find that dynamic post-training quantization hurts the downstream performance slightly while QAT achieves comparable performance to the original model. Similarly, Prato, Charlaix, and Rezagholizadeh (2020) apply QAT with STE for Transformers on neural machine translation and achieve results that are similar to the original model. I-BERT (Kim et al. 2021) eliminates floating point calculation in activation by exploiting lightweight integer-only approximation methods for non-linear operations (e.g., GELU, Softmax and LayerNorm) in BERT. The resulting I-BERT model is capable of doing pure INT8 inference thus has a better acceleration ratio.

Lower-Bit Quantization Recent works aim to push quantization to even lower precision. Lower-bit quantization faces more challenges, including difficulty to optimize, and lack of
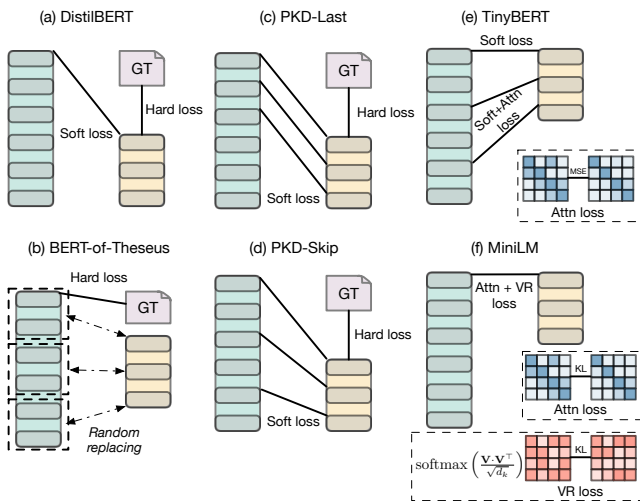
Figure 1: A summary of different KD approaches. "GT" represents ground-truth labels.

model expressivity. Shen et al. (2020) propose a group-wise quantization scheme and use second-order Hessian-based mixed-precision method (Dong et al. 2019) to quantize BERT down to 2 bits. They claim that weights corresponding to each neuron could lie in different ranges of real numbers. For example, for a multi-head self-attention module, they split the weight matrix to 12 groups, with respect to each attention head. Then they further split each group and have a total number of 128 subgroups, each of which has its own quantization range. GOBO (Zadeh et al. 2020) separates the weights into two groups — *Gaussian* and *outliers* where the former is quantized to 3 bits and the latter remains a full-precision float (FP32). TernaryBERT (Zhang et al. 2020) combines approximation-aware and loss-aware ternarization (i.e., using only $\{-1, 0, 1\}$ for weights) methods with different granularity for different components in BERT. They further add knowledge distillation to improve the performance of QAT. Bai et al. (2021) observe a large performance drop from a ternary network to a binary network when trained directly, due to its loss landscape. They propose ternary weight splitting, which initializes BinaryBERT by splitting a half-sized ternary network into two binary networks. The initialized network inherits good performance and can be further fine-tuned without optimization difficulties. Tao et al. (2022) analyze the reasons why quantization is less effective on generative LMs (e.g., GPT-2, BART). They conclude that homogeneous word embeddings caused by reduced capacity and varied distribution of weights are responsible for the failure. They propose a token-level contrastive distillation and a module-wise dynamic scaling mechanism to mitigate these two problems.

Knowledge Distillation

Knowledge Distillation (Hinton et al. 2015) is a widely used technique to transfer knowledge from a large model (teacher) to a smaller one (student) to achieve all types of efficiency. KD usually requires designing a loss function to minimize the distance of the output or intermediate features between

the student and the teacher. As illustrated in Figure 1, we summarize the designs of loss functions used in recent works distilling NLP models. Based on the loss function designs, we can further categorize the methods into logit-based KD, feature-based KD, KD with a dynamic target, and module replacing.

Logit-based KD Following Hinton et al. (2015), logit-based KD methods are the first attempts to distill a large pretrained language model into a smaller one to improve its efficiency. Logit-based KD uses the KL divergence or mean squared error (MSE) to minimize the logits between the student and the teacher. Tang et al. (2019) distills fine-tuned BERT into a BiLSTM model in a task-specific setting. The resulting BiLSTM outperforms its counterpart trained without KD by a large margin. DistilBERT (Sanh et al. 2019) distills BERT in a pretraining setting on the task of masked language modeling (MLM). The loss is a combination of three components: the original MLM loss, cosine distance, and KL divergence. After distillation, the model can be fine-tuned and perform downstream tasks. Turc et al. (2019) explore the effect of initialization for the student. They find that a student BERT pretrained with MLM outperforms random initialization and truncated teacher (Sanh et al. 2019; Sun et al. 2019) when used to initialize the student model. Liang et al. (2021) use MixUp (Zhang et al. 2018) for data augmentation to distill BERT.

Feature-based KD Instead of using only the final output, feature-based KD aims to align the intermediate features between the teacher and the student. PKD (Sun et al. 2019) introduces an MSE loss between layer representations. As shown in Figure 1(c) and (d), they propose two strategies: one aligns the student with the last few layers in the teacher (PKD-Last) and the other learns the teacher's representations in every 2 layers (PKD-Skip). The latter strategy performs slightly better in experiments. A similar technique is also presented in Aguilar et al. (2020). On the basis of PKD, TinyBERT (Jiao et al. 2020) further introduces an attention loss that aims to align the attention matrices in layers between the teacher and the student, as illustrated in Figure 1(e). TinyBERT also demonstrates that performing KD in both pretraining and fine-tuning stages can improve the performance of KD. Similarly, MiniLM (Wang et al. 2020a, 2021) aligns the attention matrix and value-value scaled dot-product (i.e., value relation loss, as shown in Figure 1(f)). The added feature complements the attention matrix (i.e., queries-keys scaled dot-product) and allows the complete transfer of multi-head self-attention. Wu, Wu, and Huang (2021) propose a multi-teacher distillation framework that use both intermediate features and soft labels from multiple teachers to distill a student and achieve better performance. DynaBERT (Hou et al. 2020) uses layer-wise KD loss to distill a teacher into a student model that has sub-networks of different widths and depths. Thus, the same model can be used on various devices with different computing budgets. MobileBERT (Sun et al. 2020) redesigns a BERT architecture that is suitable for mobile devices. In addition to the layer-wise feature distillation (Sun et al. 2019) and attention distillation (Jiao et al. 2020), they introduce a progressive knowledge transfer mechanism by distilling the model layer by layer, instead of altogether. Liu et al. (2022)

Method	Exit criterion
DeeBERT	entropy $< \theta$
RightTool	calibrated max class probability $> \theta$
FastBERT	entropy $< \theta$
RomeBERT	entropy $< \theta$
SkipBERT	max class probability $> \theta$
PABEE	patience (#consistent prediction $> \theta$)
Voting	accumulated votes $> \theta$
LeeBERT	patience (#consistent prediction $> \theta$)
Past-Future	entropy $< \theta$
PCEE-BERT	patience (#consistent IC confidence $> \theta$)
BERxiT	estimated confidence $> \theta$
CAT	estimated conformity $> \theta$

Table 2: A summary of three types of early exit methods: confidence estimation, internal ensemble, and learning to exit. θ is a predefined threshold for exiting.

exploit structural features on both token and span levels to align the student with the teacher.

Module Replacing A special case of KD is BERT-of-Theseus (Xu et al. 2020). As shown in Figure 1(b), BERT-of-Theseus does not apply any knowledge transfer loss to minimize the distance between the student and the teacher. Instead, they freeze the teacher modules and train a hybrid model by randomly replacing some modules in the teacher model with student modules. They further design a linear scheduler to increase the probability of replacement to bridge the gap between training and inference. Following this, Sparse Progressive Distillation (Huang et al. 2022) uses layer-wise KD to iteratively prune the student modules while randomly replacing each module in the teacher model with its corresponding student module with a fixed probability. After the target sparsity is hit, the replacing rate is progressively increased to 1. This method combines feature-based KD, module replacing, and pruning, achieving a super-teacher performance on GLUE (Wang et al. 2019b).

KD with Dynamic Targets In traditional KD, the teacher serves as a static target for the student to match, without any update during distillation. However, this can be suboptimal since the teacher is unaware of the student or its goal to transfer the knowledge to the student. ProKT (Shi et al. 2020) projects the supervision signals of a teacher model into the student's parameter space by decomposing the training objective into local intermediate targets with approximate mirror descent (Beck and Teboulle 2003). Zhou, Xu, and McAuley (2022) propose a simpler framework with meta learning to allow the teacher to adapt itself for better knowledge transfer. The student is evaluated on a "quiz" set after a few training steps and provides feedback to the teacher. Its first-order variant can further improve training speed and reduces memory footprint (Ma et al. 2022).

Early Exit

Early exit (EE) does not reduce the size (the total number of parameters) of the model. Instead, EE accelerates model inference by terminating inference at a particular layer based on some criteria. Although it does not make a model smaller

it can reduce computation and achieve acceleration. Early exit inserts internal classifiers (which are often simple linear layers) into a large network as triggers for early exiting. The key element in early exit methods is the exit criterion. There are three types of exit criteria: confidence estimation, internal ensemble and learning to exit. We summarize the exit criteria in Table 2.

Confidence Estimation Previous works in computer vision (Park et al. 2015; Teerapittayanon, McDanel, and Kung 2016; Kaya, Hong, and Dumitras 2019) define a metric as the proxy for confidence of prediction. The inference can exit early if the confidence reaches a threshold at an early layer. This idea is then applied to pretrained LMs (Xin et al. 2020). For each Transformer layer, a linear internal classifier (IC) is inserted after the Transformer layer. When doing inference, the model exits early when an IC outputs a predicted probability with an entropy lower than the threshold. A similar approach is proposed in RightTool (Schwartz et al. 2020). The temperature-calibrated maximum class probability is used as confidence. FastBERT (Liu et al. 2020) distills the output final classifier into earlier classifiers for better performance. Following that, RomeBERT (Geng et al. 2021) proposes gradient regularization to facilitate the KD. SkipBERT (Wang et al. 2022) replaces lower BERT layers with pre-computed representation of text chunks and uses confidence-based early exit for higher layers to achieve maximum acceleration.

Internal Ensemble A weakness of confidence estimation is poor utilization of computation. When confidence of an internal classifier fails to satisfy the exit criterion, all relevant computation becomes invalid. Reusing the results from previous layers to improve the qualify of early exit can be a promising direction. Internal ensemble approaches consider outputs and predictions from multiple internal classifiers to make better decisions. This is similar to ensemble learning, only it is within a single model.

The first work of internal ensemble, PABEE (Zhou et al. 2020), draws a comparison between overfitting in training and overthinking in inference and adapts early stopping for inference. When doing inference, the model will exit once multiple consecutive internal classifiers make the same prediction. The threshold, namely patience, is a hyperparameter that can be adjusted to achieve different trade-off between accuracy and speed. Besides improvement on performance and efficiency, PABEE achieves better adversarial robustness, which the authors attribute to the ensemble effect of internal ensemble. Sun et al. (2021) propose a diversity loss that encourages ICs to have diverse probability distributions. Then, they use a voting mechanism to internally ensemble the classifiers. Every IC has one vote in final prediction. The model will exit when one class has accumulates enough votes. Lee-BERT (Zhu 2021) promotes consistency of IC predictions by distilling them mutually. Then, it follows PABEE's patience-based strategy to decide when to exit. Liao et al. (2021) introduce a more elaborate mechanism for internal ensemble. They first train "imitation learners", which are linear layers that predict the hidden states of future layers based on hidden states that are already calculated. PCEE-BERT (Zhang et al. 2022) combines patience-based exit with confidence estimation and terminates inference when multiple layers are

Method	Optimization	Feature	Operation
PoWER	soft masking	attention	discarded
TR-BERT	RL	hidden states	forwarded
LAT	soft masking	attention	forwarded
LTP	soft masking	attention	discarded
Transkimmer	re-param.	hidden states	forwarded

Table 3: A summary of token skipping methods. This table is adapted from Guan et al. (2022).

confident.

Learning to Exit Other works use a learning-based approach to make exit decisions. BERxiT (Xin et al. 2021) trains a linear learning-to-exit (LTE) module to predict whether the current IC prediction is correct. CAT (Schuster et al. 2021) proposes a "meta consistency classifier" to predict whether the current IC prediction matches the final classifier and exits when the consistency classifier predicts a level of conformity higher than the threshold.

Token Skipping

Similar to early exit, token skipping dynamically accelerates a PLM without reducing its size. The general idea is to skip some tokens for higher layers based on their importance. A summary of these methods is shown in Table 3.

PoWER-BERT (Goyal et al. 2020) drops a portion of tokens between each Transformer layer based on the attention received by each token. The number of tokens to drop at each layer (i.e., *schedule*) is learned by jointly optimizing the sparsity of a soft mask layer with the original loss function. This approach obtains better Pareto curves of accuracy-time trade-off. TR-BERT (Ye et al. 2021) introduces a dynamic mechanism for making decisions of skipping tokens. It is trained with reinforcement learning with a reward that promotes classifier confidence and penalizes the number of retained tokens. Different from PoWER-BERT, the skipped tokens are forwarded to the last layer instead of getting removed. Length-Adaptive Transformer (LAT, Kim and Cho 2021) introduces LengthDrop that randomly skips tokens during pretraining to mitigate the gap between pretraining and fine-tuning. The schedule of LAT is searched with an evolutionary search algorithm. LTP (Kim et al. 2022) learns a threshold for each Transformer layer. Instead of following a schedule to drop a specific number of tokens, LTP simply drops tokens with a saliency score (received attention) lower than the learned threshold. Transkimmer (Guan et al. 2022) adds a skim predictor module, consisting of a small MLP and Gumbel-Softmax reparameterization before each layer. The skim predictors output a mask deciding whether to drop a token. It also employs a skim loss that optimizes the ratio of skipped tokens to the total number of tokens to encourage sparsity.

Challenges and Future Directions

Which Technique to Use? A common question asked is how to decide which technique to use in practice? Unfortunately, there is no silver bullet given that we need to take the task,
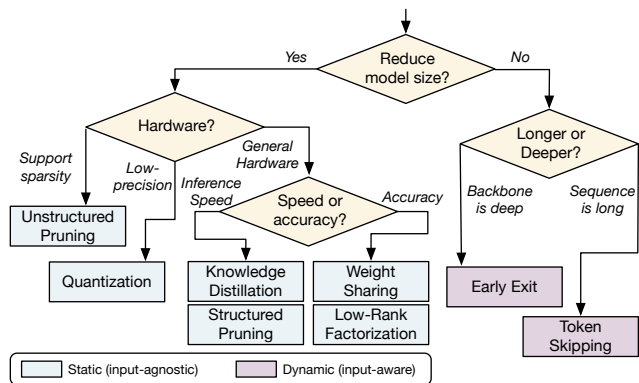

Figure 2: An *oversimplified* decision tree for model compression and acceleration.

data, backbone, and hardware into consideration. We provide an oversimplified decision tree (as shown in Figure 2) only as a starting point. Note that these techniques can often be combined for better results (to be discussed shortly).

Evaluation Although there have been benchmarks proposed for evaluating model compression and acceleration as introduced earlier, there are several drawbacks in current evaluation. First, there is no generally recognized setting for evaluation of model compression and acceleration. Different studies often yield models with different speed-up ratio, number of parameters and accuracy. Thus, it is often difficult to directly compare them, not to mention differences in hardware. Second, general NLU benchmarks like GLUE (Wang et al. 2019b) or SuperGLUE (Wang et al. 2019a) may not be the best to represent more common tasks on a mobile device. Tasks like intention detection, dense retrieval, and spam classification could be more representative.

Combining Techniques Although there have been attempts at combining multiple model compression and acceleration techniques (Kim and Awadalla 2020; Sanh, Wolf, and Rush 2020; Xu et al. 2021a), there is a lack of comprehensive and systematic study for combining compression techniques for better performance and efficiency. Constructing a best practice to compress a large model can be useful for practitioners.

Explainability and Robustness Recent works (Stanton et al. 2021; Xu et al. 2021a) cast doubt on the explainability of model compression and acceleration. Meanwhile, recent works (Du et al. 2021; Xu et al. 2021a) report negative effects of model compression on robustness. Explainable and robust compression methods can be important for applications of model compression and acceleration. Also, explainable and robust compression minimizes effort to re-evaluate the compressed model, and thus can be reliable and predictable in production (Stanton et al. 2021; Xu et al. 2021a).

Minimizing Human Effort Current compression and acceleration approaches still largely rely on human heuristics to achieve good performance. For example, knowledge distillation often requires an elaborately designed loss function; pruning relies on the saliency score; weight sharing and low-rank factorization involve expertise to appoint modules for sharing or factorization. One promising direction could be

applying Meta Learning (Finn, Abbeel, and Levine 2017) or Neural Architecture Search (Liu, Simonyan, and Yang 2019) to model compression and acceleration, to minimize the need for hyperparameters and human design.

Acknowledgments

References

Aguilar, G.; Ling, Y.; Zhang, Y.; et al. 2020. Knowledge Distillation from Internal Representations. In *AAAI*.

Bai, H.; Zhang, W.; Hou, L.; et al. 2021. BinaryBERT: Pushing the Limit of BERT Quantization. In *ACL-IJCNLP*.

Beck, A.; and Teboulle, M. 2003. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*.

Bender, E. M.; Gebru, T.; McMillan-Major, A.; and Shmitchell, S. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In *FAccT*.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Chen, P.-H.; Yu, H.-F.; Dhillon, I.; and Hsieh, C.-J. 2021. DRONE: Data-aware Low-rank Compression for Large NLP Models. In *NeurIPS*.

Chen, T.; Frankle, J.; Chang, S.; et al. 2020. The Lottery Ticket Hypothesis for Pre-trained BERT Networks. In *NeurIPS*.

Dabre, R.; and Fujita, A. 2019. Recurrent Stacking of Layers for Compact Neural Machine Translation Models. In *AAAI*.

Dehghani, M.; Gouws, S.; Vinyals, O.; et al. 2019. Universal Transformers. In *ICLR*.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

Dong, Z.; Yao, Z.; Gholami, A.; et al. 2019. HAWQ: Hessian AWare Quantization of Neural Networks With Mixed-Precision. In *ICCV*.

Du, M.; Mukherjee, S.; Cheng, Y.; et al. 2021. What do Compressed Large Language Models Forget? Robustness Challenges in Model Compression. *arXiv preprint arXiv:2110.08419*.

Edalati, A.; Tahaei, M. S.; Rashid, A.; et al. 2022. Kronecker Decomposition for GPT Compression. In *ACL*.

Fan, A.; Grave, E.; and Joulin, A. 2020. Reducing Transformer Depth on Demand with Structured Dropout. In *ICLR*.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML*, Proceedings of Machine Learning Research.

Frankle, J.; and Carbin, M. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *ICLR*.

Geng, S.; Gao, P.; Fu, Z.; and Zhang, Y. 2021. Romebert: Robust training of multi-exit bert. *arXiv preprint arXiv:2101.09755*.

Gordon, M. A.; Duh, K.; and Andrews, N. 2020. Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning. In *RepL4NLP@ACL*.

Goyal, S.; Choudhury, A. R.; Raje, S.; et al. 2020. PoWER-BERT: Accelerating BERT Inference via Progressive Word-vector Elimination. In *ICML*.

Grachev, A. M.; Ignatov, D. I.; and Savchenko, A. V. 2017. Neural Networks Compression for Language Modeling. In *PReMI*.

Guan, Y.; Li, Z.; Leng, J.; et al. 2022. Transkimmer: Transformer Learns to Layer-wise Skim. In *ACL*.

Guo, D.; Rush, A. M.; and Kim, Y. 2021. Parameter-Efficient Transfer Learning with Diff Pruning. In *ACL-IJCNLP*.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *ICLR*.

Han, X.; Zhang, Z.; Ding, N.; et al. 2021. Pre-trained models: Past, present and future. *AI Open*.

Henderson, P.; Hu, J.; Romoff, J.; et al. 2020. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*.

Hinton, G.; Vinyals, O.; Dean, J.; et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hou, L.; Huang, Z.; Shang, L.; et al. 2020. DynaBERT: Dynamic BERT with Adaptive Width and Depth. In *NeurIPS*.

Huang, S.; Xu, D.; Yen, I. E.; et al. 2022. Sparse Progressive Distillation: Resolving Overfitting under Pretrain-and-Finetune Paradigm. In *ACL*.

Jacob, B.; Kligys, S.; Chen, B.; et al. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *CVPR*.

Jiao, X.; Yin, Y.; Shang, L.; et al. 2020. TinyBERT: Distilling BERT for Natural Language Understanding. In *EMNLP (Findings)*.

Kaya, Y.; Hong, S.; and Dumitras, T. 2019. Shallow-Deep Networks: Understanding and Mitigating Network Overthinking. In *ICML*.

Kim, G.; and Cho, K. 2021. Length-Adaptive Transformer: Train Once with Length Drop, Use Anytime with Search. In *ACL-IJCNLP*.

Kim, S.; Gholami, A.; Yao, Z.; et al. 2021. I-BERT: Integer-only BERT Quantization. In *ICML*.

Kim, S.; Shen, S.; Thorsley, D.; et al. 2022. Learned Token Pruning for Transformers. In *KDD*.

Kim, Y. J.; and Awadalla, H. H. 2020. Fastformers: Highly efficient transformer models for natural language understanding. *arXiv preprint arXiv:2010.13382*.

Lacoste, A.; Luccioni, A.; Schmidt, V.; and Dandres, T. 2019. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*.

Lagunas, F.; Charlaix, E.; Sanh, V.; and Rush, A. M. 2021. Block Pruning For Faster Transformers. In *EMNLP*.

Lan, Z.; Chen, M.; Goodman, S.; et al. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *ICLR*.

Lathauwer, L. D. 2008. Decompositions of a Higher-Order Tensor in Block Terms - Part II: Definitions and Uniqueness. *SIAM J. Matrix Anal. Appl.*

Le Scao, T.; Fan, A.; Akiki, C.; et al. 2022. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model. *arXiv preprint arXiv:2211.05100*.

LeCun, Y.; Denker, J. S.; and Solla, S. A. 1989. Optimal Brain Damage. In *NIPS*.

Li, Z.; Wallace, E.; Shen, S.; et al. 2020. Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers. In *ICML*.

Liang, K. J.; Hao, W.; Shen, D.; et al. 2021. MixKD: Towards Efficient Distillation of Large-scale Language Models. In *ICLR*.

Liao, K.; Zhang, Y.; Ren, X.; et al. 2021. A Global Past-Future Early Exit Method for Accelerating Inference of Pre-trained Language Models. In *NAACL-HLT*.

Lin, Z.; Liu, J. Z.; Yang, Z.; et al. 2020. Pruning Redundant Mappings in Transformer Models via Spectral-Normalized Identity Prior. In *EMNLP (Findings)*.

Liu, C.; Tao, C.; Feng, J.; and Zhao, D. 2022. Multi-Granularity Structural Knowledge Distillation for Language Model Compression. In *ACL*.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *ICLR*.

Liu, W.; Zhou, P.; Wang, Z.; et al. 2020. FastBERT: a Self-distilling BERT with Adaptive Inference Time. In *ACL*.

Liu, X.; Sun, T.; He, J.; et al. 2021. Towards Efficient NLP: A Standard Evaluation and A Strong Baseline. *arXiv preprint arXiv:2110.07038*.

Louizos, C.; Welling, M.; and Kingma, D. P. 2018. Learning Sparse Neural Networks through L_0 Regularization. In *ICLR*.

Ma, X.; Wang, J.; Yu, L.; and Zhang, X. 2022. Knowledge Distillation with Reptile Meta-Learning for Pretrained Language Model Compression. In *COLING*.

Ma, X.; Zhang, P.; Zhang, S.; et al. 2019. A Tensorized Transformer for Language Modeling. In *NeurIPS*.

Michel, P.; Levy, O.; and Neubig, G. 2019. Are Sixteen Heads Really Better than One? In *NeurIPS*.

Min, S.; Boyd-Graber, J. L.; Alberti, C.; et al. 2020. NeurIPS 2020 EfficientQA Competition: Systems, Analyses and Lessons Learned. In *NeurIPS (Competition and Demos)*.

Mishra, A.; Latorre, J. A.; Pool, J.; et al. 2021. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*.

Nakkiran, P.; Kaplun, G.; Bansal, Y.; et al. 2020. Deep Double Descent: Where Bigger Models and More Data Hurt. In *ICLR*.

Narang, S.; Diamos, G.; Sengupta, S.; and Elsen, E. 2017. Exploring Sparsity in Recurrent Neural Networks. In *ICLR*.

Narang, S.; Undersander, E.; and Diamos, G. 2017. Block-sparse recurrent neural networks. *arXiv preprint arXiv:1711.02782*.

Noach, M. B.; and Goldberg, Y. 2020. Compressing Pre-trained Language Models by Matrix Decomposition. In *AACL-IJCNLP*.

Park, E.; Kim, D.; Kim, S.; et al. 2015. Big/little deep neural network for ultra low power inference. In *CODES+ISSS*.

Peters, M. E.; Neumann, M.; Iyyer, M.; et al. 2018. Deep Contextualized Word Representations. In *NAACL-HLT*.

Prasanna, S.; Rogers, A.; and Rumshisky, A. 2020. When BERT Plays the Lottery, All Tickets Are Winning. In *EMNLP*.

Prato, G.; Charlaix, E.; and Rezagholizadeh, M. 2020. Fully Quantized Transformer for Machine Translation. In *EMNLP (Findings)*.

Qiu, X.; Sun, T.; Xu, Y.; et al. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*.

Raffel, C.; Shazeer, N.; Roberts, A.; et al. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.*

Rajpurkar, P.; Zhang, J.; Lopyrev, K.; and Liang, P. 2016. SQuAD: 100, 000+ Questions for Machine Comprehension of Text. In *EMNLP*.

Reid, M.; Marrese-Taylor, E.; and Matsuo, Y. 2021. Subformer: Exploring Weight Sharing for Parameter Efficiency in Generative Transformers. In *EMNLP (Findings)*.

Rothe, S.; Narayan, S.; and Severyn, A. 2020. Leveraging Pretrained Checkpoints for Sequence Generation Tasks. *TACL*.

Sainath, T. N.; Kingsbury, B.; Sindhwani, V.; et al. 2013. Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets. In *ICASSP*.

Sanh, V.; Debut, L.; Chaumond, J.; and Wolf, T. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Sanh, V.; Wolf, T.; and Rush, A. M. 2020. Movement Pruning: Adaptive Sparsity by Fine-Tuning. In *NeurIPS*.

Schuster, T.; Fisch, A.; Jaakkola, T. S.; and Barzilay, R. 2021. Consistent accelerated inference via confident adaptive transformers. *arXiv preprint arXiv:2104.08803*.

Schwartz, R.; Stanovsky, G.; Swayamdipta, S.; et al. 2020. The Right Tool for the Job: Matching Model and Instance Complexities. In *ACL*.

See, A.; Luong, M.; and Manning, C. D. 2016. Compression of Neural Machine Translation Models via Pruning. In *CoNLL*.

Shen, S.; Dong, Z.; Ye, J.; et al. 2020. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. In *AAAI*.

Shi, W.; Song, Y.; Zhou, H.; et al. 2020. Learning from deep model via exploring local targets. *OpenReview preprint*.

Stanton, S.; Izmailov, P.; Kirichenko, P.; et al. 2021. Does Knowledge Distillation Really Work? *arXiv preprint arXiv:2106.05945*.

Su, D.; Zhang, H.; Chen, H.; et al. 2018. Is Robustness the Cost of Accuracy? - A Comprehensive Study on the Robustness of 18 Deep Image Classification Models. In *ECCV*.

Sun, S.; Cheng, Y.; Gan, Z.; and Liu, J. 2019. Patient Knowledge Distillation for BERT Model Compression. In *EMNLP-IJCNLP*.

Sun, T.; Zhou, Y.; Liu, X.; et al. 2021. Early Exiting with Ensemble Internal Classifiers. *arXiv preprint arXiv:2105.13792*.

Sun, Z.; Yu, H.; Song, X.; et al. 2020. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. In *ACL*.

Tahaei, M. S.; Charlaix, E.; Nia, V. P.; et al. 2021. Kronecker-bert: Learning kronecker decomposition for pre-trained language models via knowledge distillation. *arXiv preprint arXiv:2109.06243*.

Takase, S.; and Kiyono, S. 2021. Lessons on parameter sharing across layers in transformers. *arXiv preprint arXiv:2104.06022*.

Tang, R.; Lu, Y.; Liu, L.; et al. 2019. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*.

Tao, C.; Hou, L.; Zhang, W.; et al. 2022. Compression of Generative Pre-trained Language Models via Quantization. In *ACL*.

Teerapittayanon, S.; McDanel, B.; and Kung, H. T. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *ICPR*. IEEE.

Turc, I.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962*.

Vaswani, A.; Shazeer, N.; Parmar, N.; et al. 2017. Attention is All you Need. In *NIPS*.

Voita, E.; Talbot, D.; Moiseev, F.; et al. 2019. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. In *ACL*.

Wang, A.; Pruksachatkun, Y.; Nangia, N.; et al. 2019a. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. In *NeurIPS*.

Wang, A.; Singh, A.; Michael, J.; et al. 2019b. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *ICLR*.

Wang, A.; and Wolf, T. 2020. Overview of the SustaiNLP 2020 Shared Task. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*.

Wang, J.; Chen, K.; Chen, G.; et al. 2022. SkipBERT: Efficient Inference with Shallow Layer Skipping. In *ACL*.

Wang, W.; Bao, H.; Huang, S.; et al. 2021. MiniLMv2: Multi-Head Self-Attention Relation Distillation for Compressing Pretrained Transformers. In *ACL-IJCNLP (Findings)*.

Wang, W.; Wei, F.; Dong, L.; et al. 2020a. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. In *NeurIPS*.

Wang, Y.; Wang, L.; Li, V. O. K.; and Tu, Z. 2020b. On the Sparsity of Neural Machine Translation Models. In *EMNLP*.

Winata, G. I.; Madotto, A.; Shin, J.; et al. 2019. On the effectiveness of low-rank matrix factorization for lstm model compression. *arXiv preprint arXiv:1908.09982*.

Wu, C.; Wu, F.; and Huang, Y. 2021. One Teacher is Enough? Pre-trained Language Model Distillation from Multiple Teachers. In *ACL-IJCNLP (Findings)*.

Xia, M.; Zhong, Z.; and Chen, D. 2022. Structured Pruning Learns Compact and Accurate Models. In *ACL*.

Xia, Y.; He, T.; Tan, X.; et al. 2019. Tied Transformers: Neural Machine Translation with Shared Encoder and Decoder. In *AAAI*.

Xin, J.; Tang, R.; Lee, J.; et al. 2020. DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference. In *ACL*.

Xin, J.; Tang, R.; Yu, Y.; and Lin, J. 2021. BERxiT: Early Exiting for BERT with Better Fine-Tuning and Extension to Regression. In *EACL*.

Xu, C.; Zhou, W.; Ge, T.; et al. 2020. BERT-of-Theseus: Compressing BERT by Progressive Module Replacing. In *EMNLP*.

Xu, C.; Zhou, W.; Ge, T.; et al. 2021a. Beyond Preserved Accuracy: Evaluating Loyalty and Robustness of BERT Compression. In *EMNLP*.

Xu, J.; Zhou, W.; Fu, Z.; et al. 2021b. A Survey on Green Deep Learning. *arXiv preprint arXiv:2111.05193*.

Ye, D.; Lin, Y.; Huang, Y.; and Sun, M. 2021. TR-BERT: Dynamic Token Reduction for Accelerating BERT Inference. In *NAACL-HLT*.

Yuan, M.; and Lin, Y. 2006. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*.

Zadeh, A. H.; Edo, I.; Awad, O. M.; et al. 2020. GOBO: Quantizing Attention-Based NLP Models for Low Latency and Energy Efficient Inference. In *MICRO*.

Zafrir, O.; Boudoukh, G.; Izsak, P.; and Wasserblat, M. 2019. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*.

Zhang, H.; Cissé, M.; Dauphin, Y. N.; and Lopez-Paz, D. 2018. mixup: Beyond Empirical Risk Minimization. In *ICLR*.

Zhang, M. S.; and Stadie, B. C. 2020. One-Shot Pruning of Recurrent Neural Networks by Jacobian Spectrum Evaluation. In *ICLR*.

Zhang, W.; Hou, L.; Yin, Y.; et al. 2020. TernaryBERT: Distillation-aware Ultra-low Bit BERT. In *EMNLP*.

Zhang, Z.; Zhu, W.; Zhang, J.; et al. 2022. PCEE-BERT: Accelerating BERT Inference via Patient and Confident Early Exiting. In *NAACL-HLT (Findings)*.

Zhou, W.; Xu, C.; Ge, T.; et al. 2020. BERT Loses Patience: Fast and Robust Inference with Early Exit. In *NeurIPS*.

Zhou, W.; Xu, C.; and McAuley, J. J. 2022. BERT Learns to Teach: Knowledge Distillation with Meta Learning. In *ACL*.

Zhu, W. 2021. LeeBERT: Learned Early Exit for BERT with cross-level optimization. In *ACL-IJCNLP*.