Magic Pyramid: Accelerating Inference with Early Exiting and Token Pruning

Xuanli He[1] **Iman Keivanloo**[2] **Yi Xu**[2] **Xiang He**[2]

Belinda Zeng[2] **Santosh Rajagopalan** [2] **Trishul Chilimbi**[2]

[1]Monash University, Australia [2]Amazon
xuanli.he1@monash.edu, {imankei,yxaamzn}@amazon.com

Abstract

Pre-training and then fine-tuning large language models is commonly used to achieve state-of-the-art performance in natural language processing (NLP) tasks. However, most pre-trained models suffer from low inference speed. Deploying such large models to applications with latency constraints is challenging. In this work, we focus on accelerating the inference via conditional computations. To achieve this, we propose a novel idea, Magic Pyramid (MP), to reduce both width-wise and depth-wise computation via token pruning and early exiting for Transformer-based models, particularly BERT. The former manages to save the computation via removing non-salient tokens, while the latter can fulfill the computation reduction by terminating the inference early before reaching the final layer, if the exiting condition is met. Our empirical studies demonstrate that compared to previous state of arts, MP is not only able to achieve a speed-adjustable inference, but also to surpass token pruning and early exiting by reducing up to 70% giga floating point operations (GFLOPs) with less than 0.5% accuracy drop. Token pruning and early exiting express distinctive preferences to sequences with different lengths. However, MP is capable of achieving an average of 8.06x speedup on two popular text classification tasks, regardless of the sizes of the inputs.

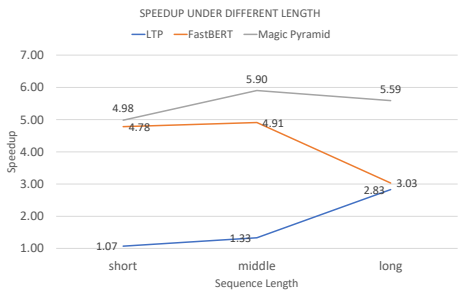
1 Introduction

In the past few years, owing to the success of Transformer-based [20] pre-trained models, such as BERT [1], RoBERTa[11], GPT2[13], *etc.,* we have experienced a performance breakthrough in natural language processing (NLP) tasks. With a small amount of fine-tuning, the pre-trained models can achieve state-of-the-art performance across different tasks [1, 11, 13]. Nevertheless, the outperforming models are evaluated in offline settings, and the inference latency is not assessed or considered as a quality factor.

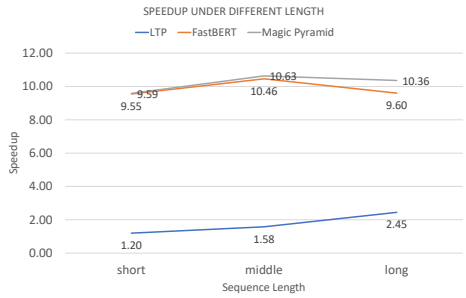
However, adapting and deploying such large pre-trained models to production systems (*e.g.,* online shopping services) is not straightforward due to the latency constraint and the large volume of incoming requests (*e.g.,* millions of requests per second). Prior to this work, researchers have proposed to compress a large model via either model pruning [12, 2, 6] or token pruning [22, 3, 9]. In addition, compressing a large teacher model into a compact model via knowledge distillation has been studied extensively in the past [14, 17, 18, 7, 4]. Finally, another line of work targets on plugging multiple sub-classifiers into deep neural networks to enable a flexible computation on demand, *a.k.a.,* early exiting [19, 8, 15, 10]

The token pruning concentrates on a width-wise computational reduction, whereas the early exiting succeeds in a depth-wise inference acceleration. Our study shows that for certain tasks where the

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.



(a) Yelp.



(b) AG news

Figure 1: Speedup of LTP (token pruning), FastBERT (early exiting) and MP (ours) under different sequence lengths on Yelp and AG news. X axis is sequence length, while Y axis is speedup. short: 1-35 tokens; middle: 35-70 tokens; long: >70 tokens

input data is diverse (in terms of sequence length), these two latency reduction methods perform in the opposite direction. As illustrated in Figure 1 (a), speedup (Y axis) achieved via early exiting (FastBERT) decreases for long input sizes (X axis). However, token pruning (LTP) speedup rises as the input size increases. We believe these two approaches are orthogonal and can be combined into a single model to maintain the latency reduction gain across the variable input length. In this work, we present a novel approach, *Magic Pyramid* (MP), to encourage a speed-adjustable inference. The contribution of this paper includes:

- Our empirical study shows that token pruning and early exiting are potentially orthogonal. This motivates further research on employing the two orthogonal inference optimization methods within a single model for model inference acceleration.
- We propose a method (referred to as Magic Pyramid) to exploit the synergy between token pruning and early exiting and attain higher computational reduction from width and depth perspectives.
- Compared to two strong baselines, our approach can significantly accelerate the inference time with an additional 0.5-2x speedup but less than 0.5% degradation on accuracy across five classification tasks.

2 Related Work

Large pre-trained models have demonstrated that increasing the model capacity can pave the way for the development of superior AI. However, as we have limited resources allocated for production systems, there has been a surge of interest in efficient inference. Previous works [14, 17, 18, 7, 4] have opened a window into an effective model compression via knowledge distillation (KD) [5]. The core of KD is to use a compact student model to mimic the behavior or structure of a large teacher model. As such, the performance of the student model is as accurate as its teacher, but consuming less computation.

Other researchers approach efficient inference by manipulating the original model. One elegant solution is pruning, which can reduce the computation by removing non-essential components. These components can be either model parameters (model pruning) [12, 2, 6] or tokens (token pruning) [22, 3, 9]. In addition, one can boost the speed of the numerical operations of a model through quantization [24, 23, 16].

The aforementioned works lack flexibility in terms of the speedup, albeit some success. To satisfy varying demands, we have to train multiple models. Since deep neural networks can be considered as a stack of basic building blocks, a list of works introduces early exiting [19, 8, 15, 10], which attaches a set of sub-classifiers to these sub-networks to encourage an adjustable inference within a single model, when needed. As opposed to the prior works, which focus on the one-dimensional speedup, this work takes the first step to superimpose token pruning on early exiting. Our empirical studies confirm that these two approaches can accelerate the inference collaboratively and significantly.

3 Methodology - Proposed Method

Prior to this work, token pruning and early exiting have been proven to be effective in accelerating the inference [22, 3, 15, 10, 9]. However, as shown in Figure 1, these approaches fall short of reducing the latency at two ends, *i.e.,* short sequences and long sequences. For example, Figure 1 (a) shows that LTP (token-pruning) provides the highest speed-up for long input sequences. While FastBERT (early exiting) speedup drops as the input size increases from short to long. Therefore we propose a novel approach: *Magic Pyramid* (MP), which benefits from a combination of token pruning and early exiting. Figure 2 provides a schematic illustration of MP. First of all, MP enables to terminate an inference at any layer when needed. Second, with the increase of the depth of Transformer, redundant tokens can be expelled. The detailed designs are provided in the rest of this section.

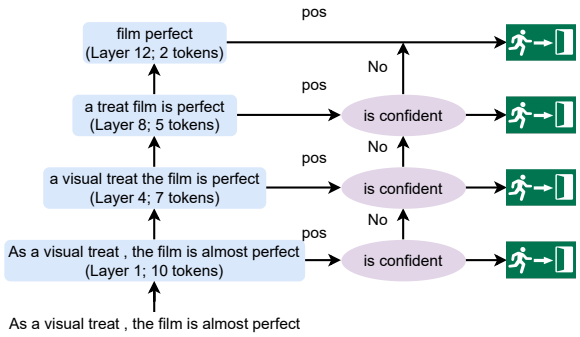


Figure 2: Schematic illustration of magic pyramid for a sentiment analysis task

Transformer architecture Owing to it outstanding performance, Transformer [20] has become a de facto model for NLP tasks, especially after the triumph of pre-trained language models [1, 11, 13]. A standard Transformer is comprised of L stacked Transformer blocks: $\{\mathrm{B}_l\}_{l=1}^{L}$, where each block B_l is formulated as:

$$H'_l = \mathrm{MultiHead}(H_{l-1}) \tag{1}$$

$$H_l = \mathrm{FFN}(H'_l) \tag{2}$$

where $H \in \mathbb{R}^{n\times d}$ and $H' \in \mathbb{R}^{n\times d}$ are the hidden states. n is sequence length, while d is the feature dimension. $\mathrm{MultiHead}$ and FFN are multi-head attention module and position-wise feedforward module respectively. We omit $\mathrm{Residual}$ module and $\mathrm{Layernorm}$ module in between for simplicity.

Early exiting As shown in Figure 2, in addition to the Transformer backbone and a main classifier, one has to attach an individual sub-classifier module ($\mathrm{subclassifier}(\cdot)$) to the FFN of each Transformer block B_i. As such, one can choose to terminate the computation at any layer, when a halt value τ is reached.

Following [10], the $\mathrm{subclassifier}(\cdot)$ consists of a Transformer block B, a pooling layer Pooler and a projection layer $\mathrm{Projector}$ with a $\mathrm{softmax}$ function. Pooler extracts the hidden states of $[\mathrm{CLS}]$ as the representation of the input, while $\mathrm{Projector}$ projects the dense vector into N-class logits.

Similar to [10], we leverage a two-stage fine-tuning to enhance the performance of sub-classifiers via knowledge distillation. Specifically, we first train the Transformer backbone and the primary classifier through a standard cross entropy between the ground truth y and the predictions y'. Afterward, we freeze the backbone and the primary classifier, but train each $\mathrm{subclassifier}(\cdot)$ via a Kullback–Leibler divergence:

$$D_{KL}(p_s, p_t) = \sum_{i=1}^{N} p_s(i) * \log \frac{p_s(i)}{p_t(i)} \tag{3}$$

where p_s and p_t are the predicted probability distribution from the $\mathrm{subclassifier}(\cdot)$ and the main classifier respectively. Since there are $L-1$ sub-classifiers, the loss of the second stage can be formulated as:

$$\mathcal{L}(p_{s_1}, ..., p_{s_{L-1}}, p_t) = \sum_{l=1}^{L-1} D_{KL}(p_{s_i}, p_t) \tag{4}$$

Once all modules are well-trained, we can stitch them together to achieve a speed-adjustable inference. At each layer l, we first obtain the hidden states H_l from the Transformer block B_l. Then a probability p_{s_l} can be computed from $\mathrm{subclassifier}(H_l)$. One can use p_{s_l} to calculate the uncertainty u_l via:

$$u_l = \frac{\sum p_{s_l} \log p_{s_l}}{\log \frac{1}{N}} \tag{5}$$

where u_l is bound to $\{0, 1\}$. If $u_l \leq \tau$, we can terminate the computation. A larger τ suggests a faster exit.

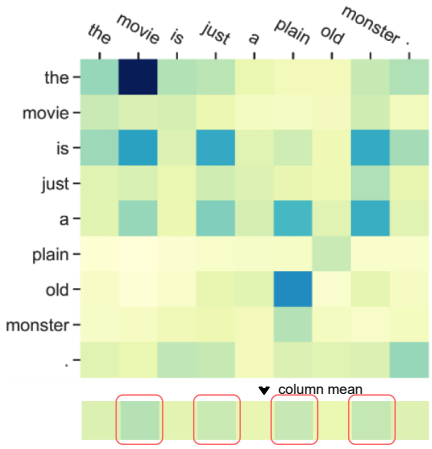


Figure 3: An example of attention probability in a single head. Darker color suggests a higher attention score. The bottom heatmap is the column mean of the attention matrix.

Token pruning The core of the Transformer block is the $\mathrm{MultiHead}$ module, which is responsible for a context-aware encoding of each token. Notably, we compute pairwise importance among all tokens within the input via self-attention. The attention score of each head h between x_i and x_j is obtained from:

$$A^h(x_i, x_j) = \mathrm{softmax}(\frac{H(x_i)^T \mathbf{W}_q^T \mathbf{W}_k H(x_j)}{\sqrt{d}}) \tag{6}$$

where $H(x_i) \in \mathbb{R}^d$ and $H(x_j) \in \mathbb{R}^d$ are the hidden states of x_i and x_j respectively. $\mathbf{W}_q \in \mathbb{R}^{d_h \times d}$ and $\mathbf{W}_k \in \mathbb{R}^{d_h \times d}$ are learnable parameters. d_h is set to d/N_h, and N_h is the number of heads used in the Transformer block B. Since we have to conduct n^2 such operation to acquire an attention score matrix $A \in \mathbb{R}^{n \times n}$, the complexity of $\mathrm{MultiHead}$ quadratically scales with the sequence length. Therefore, we encounter a computational bottleneck when working on long sequences. However, if we take the average of A along the ith column, we notice that the different tokens have distinctive scores as shown in Figure 3. Tokens with large scores tend to be more salient than others, as they receive more attention. As such, we can prune the non-salient tokens to save the computation. We formally define an importance score of each token x_i at layer l as:

$$s^l(x_i) = \frac{1}{N_h} \frac{1}{n} \sum_{h=1}^{N_h} \sum_{j=1}^{n} A_l^h(x_i, x_j) \tag{7}$$

Before this work, researchers have proposed two approaches to remove the unimportant tokens based on $s^l(\cdot)$: *i)* top-k based pruning and *ii)* threshold-based pruning [3, 22, 9]. In this work, we follow [9], which leverages a layer-wise learnable threshold to achieve a fast inference and is superior to other works[22, 3]. We first fine-tune Transformer parameters Θ on a downstream task. Then we introduce a two-stage pruning scheme to seek a suitable threshold Δ and Θ, which can accelerate the inference and maintain a decent accuracy. In the first pruning stage, we apply a gating function $\sigma(\cdot)$ to weight the outputs H_l from the current layer l, before we pass them to the next layer $l + 1$ like this:

$$M^l(x_i) = \sigma(\frac{s^l(x_i) - \Delta^l}{T}) \tag{8}$$

$$\hat{H}_l(x_i) = H_l(x_i) \otimes M^l(x_i) \tag{9}$$

where σ is a sigmoid function, $\otimes$ is an element-wise multiplication, T is a temperature parameter and $\Delta^l \in \mathbb{R}$ is a learnable threshold at layer l. If $M^l(x_i)$ approaches zero, $\hat{H}_l(x_i)$ will become zero as well. As such, $\hat{H}_l(x_i)$ has no impact on the subsequent layers. At this stage, since $\sigma(\cdot)$ allows the flow of the back-propagation, both Θ and Δ can be optimized. In addition, [9] also impose a $L1$ loss on M as a regularizer to encourage the pruning operation. Please refer to their paper for the details.

In the second pruning stage, we binarize the mask values at the inference time via:

$$M^l(x_i) = \begin{cases} 1, & s^l(x_i) - \Delta^l > 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

If $s^l(x_i)$ is below the threshold Δ^l, x_i is subject to the removal from layer l and will not contribute towards the final predictions. We freeze Δ but update Θ, such that the model can learn to accurately predict the labels merely conditioning on the retained tokens.

4 Experiments

To examine the effectiveness of the proposed approach, we use five language understanding tasks as the testbed. We describe our datasets and our experimental setup in the following.

Data	Train	Test	Task
AG news	120K	7.6K	topic
Yelp	560K	38K	sentiment
QQP	364K	40K	paraphrase
MRPC	3.7K	408	paraphrase
RTE	2.5K	277	language inference

Table 1: The statistics of datasets

Datasets The first two tasks are: *i)* AG news topic identification [25], and *ii)* Yelp polarity sentiment classification [25]. The last three are: *i)* Quora Question Pairs (QQP) similarity detection dataset, *ii)* Microsoft Research Paraphrase Corpus (MRPC) dataset and *iii)* Recognizing Textual Entailment (RTE) dataset. All datasets are from GLUE benchmark [21] and focus on predicting between a sentence pair. The datasets are summarized in Table 1.

Datasets	BERT	DistilBERT	LTP	FastBERT	MP (ours)
AG news	3,-,-,-	3,-,-,-	3,1,2,-	3,-,-,2	3,1,2,2
Yelp	3,-,-,-	3,-,-,-	3,1,2,-	3,-,-,2	3,1,2,2
QQP	5,-,-,-,	5,-,-,-,	5,2,5,-	5,-,-,5	5,2,5,5
MRPC	10,-,-,-,	10,-,-,-,	10,10,5,-	10,-,-,5	10,10,5,5
RTE	10,-,-,-,	10,-,-,-,	10,10,5,-	10,-,-,5	10,10,5,5

Table 2: The number of epochs used for regular training, soft pruning, hard pruning, subclassifiers training on different datasets. "-" indicates the corresponding stage is inactive.

Experimental setup We compare our approach with four baselines: *i)* standard BERT [1], *ii)* distilBERT [14], *iii)* learned token pruning (LTP) [9] and *iv)* FastBERT [10]. Except distilBERT, all approaches are fine-tuned on uncased BERT-base model (12 layers).

For training, we use a batch size of 32 for QQP, MRPC, and RTE. We set this to 64 for AG news and Yelp. Since different approaches adopt different training strategies, we unify them as four steps:

1. Regular training: training a model Θ without additional components;
2. Soft pruning: training a model Θ and threshold Δ;
3. Hard pruning: training a model Θ with the binarized mask values;
4. Sub-classifiers training: training sub-classifiers on Equ. (4). For MP, we also activate the pruning operations.

We report the number of the training epochs of different steps for all approaches in Table 2. Similar to [9], we vary the threshold of the final layer Δ^L from 0.01 to 0.08, and the threshold for Δ^l is set to

$\Delta^L l/L$. We search the temperature T in a search space of {1e−5,2e−5,5e−5} and vary λ from 0.001 to 0.2. We use a learning rate of 2e−5 for all experiments. We consider accuracy for the classification performance and giga floating point operations (GFLOPs) for the speedup.

	AG news		Yelp		QQP		MRPC		RTE	
	Acc.	GFLOPs	Acc.	GFLOPs	Acc.	GFLOPs	Acc.	GFLOPs	Acc.	GFLOPs
BERT	94.3	9.0 (1.00x)	95.8	17.2 (1.00x)	91.3	5.1 (1.00x)	85.3	9.2 (1.00x)	68.6	11.2 (1.00x)
distilBERT	94.4	4.5 (2.00x)	95.7	8.6 (2.00x)	90.4	2.6 (2.00x)	84.6	4.6 (2.00x)	58.8	5.6 (2.00x)
LTP	94.3	5.3 (1.72x)	94.7	7.4 (2.32x)	90.6	3.2 (1.60x)	84.8	6.2 (1.48x)	67.8	7.5 (1.50x)
FastBERT	94.3	2.3 (3.97x)	94.8	2.8 (6.18x)	90.7	1.6 (3.20x)	84.3	4.3 (2.13x)	67.6	8.4 (1.33x)
MP (ours)	94.3	1.8 (4.95x)	94.5	2.1 (8.25x)	90.4	1.3 (4.03x)	83.8	3.3 (2.77x)	67.5	6.5 (1.72x)

Table 3: The accuracy and GFLOPs of BERT [1], distilBERT [14], LTP (learned token pruning) [9], FastBERT [10] and MP (ours) on different datasets. The numbers in parentheses are speedup.

	AG news			Yelp		
τ	0.1	0.5	0.8	0.1	0.5	0.8
FastBERT	3.97x	10.30x	11.95x	3.15x	6.18x	8.84x
MP (ours)	4.95x	10.53x	11.95x	5.35x	8.25x	10.10x

Table 4: Speedup of FastBERT and MP with different τ.

For token pruning approach, previous works [22, 9] have shown that there exits a trade-off between accuracy and speedup. Thus, we report the performance of models achieving smallest GFLOPs with at most 1% accuracy drop compared to the BERT baseline. Similarly, the speedup of FastBERT is also controlled by the halt value τ. We select τ obtaining a on-par accuracy with the token pruning competitors for the sake of a fair comparison. This selection criterion is applied to MP as well.

Table 3 demonstrates that all approaches experience loss in accuracy, when a fast inference is activated. Overall, FastBERT is superior to disilBERT and LTP in terms of both accuracy and GFLOPs. Under the similar accuracy, our approach manages to have a significantly faster inference than FastBERT, which leads to up to 2.13x extra speedup. We notice that the speedup and accuracy also correlate to the complexity of tasks and the number of training data. Specifically, for the sentence-pair classification tasks, since QQP has much more data (*c.f.,* Table 1), it achieves 4.03x speedup with a loss of 1% accuracy. On the contrary, RTE and MRPC obtain at most 2.77x speedup with the same amount of accuracy degradation. Under the same magnitude of the training data, as AG news and Yelp are simpler than QQP, they can gain up to 8.25x speedup after sacrificing 1% accuracy.

Gains over FastBERT In section 3, we have claimed that MP can benefit from both token pruning and early exiting. Although this claim is evidenced in Table 3, we are interested in investigating whether such gains consistently hold, when tuning τ to control the speed of the inference. According to Table 4, MP can drastically boost the speedup of FastBERT, except for an aggressive τ, which will cause the computation to terminate at the first two layers.

Speedup on sequences with different lengths Intuitively, longer sentences tend to have more redundant tokens, which can confuse the lower sub-classifiers. Consequently, longer sentences require more computation before reaching a lower uncertainty u. We bucket the Yelp and AG news dataset into three categories: *i)* short sequences (1-35 tokens), *ii)* middle sequences (35-70 tokens) and *iii)* long sequences (>70 tokens). Figure 1 indicates that LTP prefers long sequences, while FastBERT favors short sequences. Since MP combines the early exiting with the token pruning, it can significantly accelerate both short and long sequences, compared to the two baselines.

5 Conclusion

In this work, we introduce Magic Pyramid, which can maintain a trade-off between speedup and accuracy for BERT-based models. Since MP is powered by two outstanding efficiency-encouraging approaches, it can yield substantially faster inference over the baselines up to additional 2x speedup. We also found that token pruning and early exiting falls to efficiently handle sequences under certain length groups. In contrary, such limitations can be combated by MP, thereby our approach can indiscriminately accelerate inference for every input data (i.e, inference request) regardless of its length.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[2] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2019.

[3] Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pages 3690–3699. PMLR, 2020.

[4] Xuanli He, Islam Nassar, Jamie Kiros, Gholamreza Haffari, and Mohammad Norouzi. Generate, annotate, and learn: Generative models advance self-training and knowledge distillation. *arXiv preprint arXiv:2106.06168*, 2021.

[5] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[6] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33, 2020.

[7] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 4163–4174, 2020.

[8] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3301–3310. PMLR, 09–15 Jun 2019.

[9] Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Joseph Hassoun, and Kurt Keutzer. Learned token pruning for transformers. *arXiv preprint arXiv:2107.00910*, 2021.

[10] Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. Fastbert: a self-distilling bert with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6035–6044, 2020.

[11] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[12] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in Neural Information Processing Systems*, 32:14014–14024, 2019.

[13] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

[15] Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A Smith. The right tool for the job: Matching model and instance complexities. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6640–6651, 2020.

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

[17] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 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)*, pages 4323–4332, 2019.

[18] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, 2020.

[19] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469, 2016.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[21] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 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*, pages 353–355, 2018.

[22] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE, 2021.

[23] Krzysztof Wróbel, Michał Karwatowski, Maciej Wielgosz, Marcin Pietroń, and Kazimierz Wiatr. Compression of convolutional neural network for natural language processing. *Computer Science*, 21(1), 2020.

[24] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert.

[25] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28:649–657, 2015.