Efficient Inference of Transformers in Natural Language Processing: Early Exiting and Beyond

by

Ji Xin

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2023

Examining Committee Membership

The following serve on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	**Davood Rafiei** Professor, Computer Science University of Alberta
Supervisors:	**Jimmy Lin** Professor, Computer Science University of Waterloo
	Yaoliang Yu Associate Professor, Computer Science University of Waterloo
Internal Member:	**Charles Clarke** Professor, Computer Science University of Waterloo
	Wenhu Chen Assistant Professor, Computer Science University of Waterloo
Internal-External Member:	**Lukasz Golab** Professor, Management Sciences University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The thesis includes contributions from five published papers and public manuscripts. Ji Xin is the first author of all five articles, and is responsible for most parts of implementation, experimentation, and writing. Other authors include:

- Chapter 3 (Xin et al., 2020a): Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin.
- Chapter 4 (Xin et al., 2021a): Raphael Tang, Yaoliang Yu, and Jimmy Lin.
- Chapter 5 (Xin et al., 2020b): Rodrigo Nogueira, Yaoliang Yu, and Jimmy Lin.
- Chapter 6 (Xin et al., 2021b): Raphael Tang, Yaoliang Yu, and Jimmy Lin.
- Chapter 7 (Xin et al., 2022): Raphael Tang, Zhiying Jiang, Yaoliang Yu, and Jimmy Lin.

They are mainly responsible for brainstorming ideas, discussing experimental details, and proofreading drafts.

Abstract

Large-scale pre-trained transformer models such as BERT have become ubiquitous in Natural Language Processing (NLP) research and applications. They bring significant improvements to both academia benchmarking tasks and industry applications: the average score on the General Language Understanding Evaluation benchmark (GLUE) has increased from 74 to $90+$; commercial search engines such as Google and Microsoft Bing are also applying BERT-like models to search.

Despite their exciting power, these increasingly large transformer-based models are notorious for having billions of parameters and being slow in both training and inference, making deployment difficult when inference time and resources are limited. Therefore, model efficiency has become a more important and urgent problem in the transformer era. In this thesis, we propose and innovate methods for efficient NLP models. We choose to specifically focus on inference efficiency: pre-trained models are almost always publicly available, and fine-tuning is performed on relatively small datasets without strict time constraints; inference, by contrast, needs to be performed repetitively and typically in a real-time setting.

First, we propose the early exiting idea for transformers. Considering that the transformer model has multiple layers with identical structures, we try to reduce the number of layers used for inference by dynamic early exiting. During inference, if an intermediate transformer layer predicts an output of high confidence, we directly exit from this layer and use the current output as the final one. We apply the early exiting idea on sequence classification tasks and show that it is able to greatly improve inference efficiency.

We then explore a few extensions to the early exiting idea: (1) early exiting for low-resource datasets—in this case, the straightforward fine-tuning methods fail to train the model to its full potential and we propose a method to better balance all layers of the model; (2) early exiting for regression datasets—in this case, the output is no longer a distribution where we can directly estimate confidence, and we design a learning-to-exit module to explicitly learn confidence estimation; (3) early exiting for document reranking—in this case, the two classes that the model tries to distinguish are highly asymmetric and we design an asymmetric early exiting method to better handle this task. We also extend early exiting to another direction—selective prediction. In this setting, if we have low confidence in the final prediction, we abstain from making predictions

at all. We propose better ways for confidence estimation and also discuss a few applications for selective prediction.

Finally, we discuss the combination of multiple efficiency methods, including early exiting itself and other popular methods such as distillation, pruning, quantization, etc. We propose a conceptual framework to treat each efficiency method as an operator. We conduct experiments to show interesting properties of these operators when they combine, which provide useful guidelines for designing and evaluating the application of combining multiple efficiency methods.

The thesis presents a series of modeling and experimental contributions for efficient transformer models. We not only largely reduce the inference time for many NLP and IR applications, but also provide insights to understand the efficiency problem from a novel perspective.

Acknowledgments

Thanks to the existence of acknowledgments in a thesis, Ph.D. students can, after all the years of training in rigor and precision, finally write something subjective or even emotional.

I'd like to thank my advisors, Jimmy and Yaoliang, for everything they have done for me. They walked me through all aspects of research. More importantly, after hearing countless stories about disagreements between advisors and students, I was pleasantly surprised to find that the advising styles of Jimmy and Yaoliang were exactly what I had hoped for. Not only did they give me a lot of freedom to explore the field at my preferred pace and direction, but they also provided me with enough guidance to keep me on the right track, especially when I got lost. One thing that I really appreciate is that compared to momentary achievements (e.g., publishing a certain paper), they care more about the eventual growth and success of students. It's because of my advisors that these years have been busy but fulfilling and rewarding.

I'd also like to thank the rest of my thesis committee, Charlie, Lukasz, Wenhu, and Davood, for patiently reading through my gibberish and providing all the valuable suggestions.

My internship at Microsoft was intense and educational. I'd like to thank my mentor, Chenyan, who showed me the level of passion and dedication it takes to become a first-class researcher.

The journey would have been gloomy and arduous were it not for the support from my collaborators and friends. Thanks Ralph, for being the second author in many of my papers; I'm glad you make it through the great challenge. Thanks Achyudh, Brandon, Mavis, Gin, Rodrigo, Crystina, Jiarui, Wei, Xueguang, Minghan, Peng, Leo, and Linqing, for the memorable hours in and out of the DSG lab, working and slacking. Thanks Haotian, Victor, and Michael, for putting up with my silly questions. Thanks Zeou, Guojun, Kaiwen, Allen, Jingjing, and Alix, for the fun we had. Thanks Yao, Aoqian, Haoyu, Yuan, Bo, Yangtian, and Kaisong, for being around in my spare time, especially enduring my awful singing at karaoke.

Special thanks to my old friends from the undergrad lab—Zhiyuan Liu, Luming Tang, and Xu Han—for all the compassionate encouragement.

Finally, I owe a debt of the deepest gratitude to my family—my wife, my mom, and my late father. The company of my wife has been the source of courage and peace of mind. As we jokingly but precisely describe it, the best part of our relationship is that it offers double the

joy and half the sorrow. As for my parents, I simply lose track of how much they have done and sacrificed for me. Above all, I'm grateful for the happy childhood they provided, a period that nurtured me into who I am today. I wanted to conclude with the cliché *I want to make you proud*, but come to think of it, my parents have already been proud of me all the time. It's such a bittersweet memory that in my dad's final days, one of his few happy moments was when I told him I would complete the Ph.D. degree ahead of schedule. It reminds me that when I was a kid, dad and I would sit on the balcony after dinner looking up at the stars, and one day he laughed when I said I wanted to become a scientist (more precisely, an astronomer) when I grew up. Today's degree might be the closest I've come to being a scientist, but I finally realize that dad would, additionally, be prouder and more content to see me living a happy and peaceful life. I shall not disappoint him.

Dedication

To my family and all the happy memories.

Contents

List of Tables xiii

List of Figures xv

1 Introduction 1

1.1 Challenges . . . 2

1.2 Early Exiting and Beyond . . . 3

1.3 Contributions . . . 7

2 Background 9

2.1 Model Architecture: Transformers and Training . . . 9

2.2 Related Work: Efficiency Methods for Transformers . . . 13

2.3 Experimental Setup: Datasets, Implementation, and Pre-trained Checkpoints . . . 19

3 Early Exiting for Sequence Classification 22

3.1 Overview . . . 22

3.2 Modeling Details . . . 23

3.3 Experimental Setup . . . 26

3.4 Experimental Results . . . 26

3.5 Summary . . . 32

4 Early Exiting for Sequence Regression and Low-Resource Datasets **33**

4.1 Overview 34

4.2 Model Structure and Fine-Tuning 35

4.3 Exiting Decision Making 37

4.4 Experimental Setup 38

4.5 Experimental Results 41

4.6 Summary 50

5 Early Exiting for Document Reranking **51**

5.1 Overview 51

5.2 Early Exiting for Document Ranking 52

5.3 Experimental Setup 55

5.4 Experimental Results 56

5.5 Summary 60

6 Selective Prediction for NLP Tasks **61**

6.1 Overview 61

6.2 Background 63

6.3 Error Regularization 66

6.4 Experimental Setup 67

6.5 Experimental Results 68

6.6 Summary 76

7 Commutativity and Cumulativeness of Efficiency Operators **77**

7.1 Overview 77

7.2 Modeling Details for Operators 78
7.3 Experimental Design 81
7.4 Operator Commutativity and Order 82
7.5 Operator Cumulativeness and Predictability of Pipelines 85
7.6 Summary 89

8 Conclusion and Future Work **90**

References **95**

List of Tables

2.1 Hyper-parameters used in a typical transformer ($\text{BERT}_{\text{BASE}}$). 11

2.2 Statistics of NLP datasets used in the thesis. STS-B and SICK are regression datasets, thus # Classes = 1. 21

3.1 Comparison between baseline (original BERT/RoBERTa), DeeBERT, and other acceleration methods. LayerDrop only reports results on SST-2 and MNLI. Time savings of DistilBERT and LayerDrop are estimated by reported model size reduction. 27

4.1 Inference runtime in seconds for each model and dataset. 39

4.2 Test set results comparing baselines (raw $\text{BERT}_{\text{BASE}}$/$\text{BERT}_{\text{LARGE}}$, from the original paper), DistilBERT, and early exiting with *Alternating* fine-tuning. Metric for model quality: score for RAW baselines and relative scores for others. Metric for model efficiency: total layers for RAW; relative saved layers for others (w.r.t. raw models). 45

4.3 Comparing LTE with PABEE on STS-B. 47

5.1 MS MARCO passage development set results. MB: MonoBERT; eeMB: early exiting MonoBERT. 56

5.2 ASNQ development set results. CT: Cascade Transformer; eeMB: early exiting MonoBERT. Absolute values of nDCG and MRR scores are *not directly comparable* between CT and eeMB, due to differences in dataset preparation. . . 56

6.1 Choices for λ for different models and regularization methods. 69

6.2 Comparing selective prediction performance of different models and confidence estimators. All metrics except AUC are in percentages. $\uparrow$: higher is better; $\downarrow$: lower is better. 69

6.3 Comparing different regularizers (Reg.) for different models and datasets. Selective prediction performance is measured by AUC and RPP. All metrics except AUC are in percentages. 71

6.4 Dataset statistics. bMNLI/bSST-5 are binarized version of MNLI/SST-5, with two normal labels and a special *no-answer* label. 73

6.5 Selective prediction performance of different models and regularization methods (Reg.) on two datasets with the *no-answer* label. All metrics except AUC are in percentages. 74

7.1 The mean and the standard deviation (SD) of *distances* between trade-off curves belonging to same/different orders (the same ordering is run with multiple random seeds). For all entries, the 1-SD intervals of same/different orders overlap. . . . 85

7.2 Accuracy drops and time savings provided by quantization (**Q**) and dynamic length inference (**L**) applied at the end of pipelines. The accuracy drops and time savings of most operators are cumulative. 86

List of Figures

1.1 Early exiting model overview. There are n layers in total; blue blocks are transformer layers and orange blocks are classifiers. This diagram will be used in Chapter 3 and Chapter 5. 4

2.1 The model structure of a transformer layer (left) and the multi-head attention mechanism (right).[1] 10

2.2 Diagrams for efficiency methods. The model consists of an *embedding layer* (EMB) at the bottom, a few *transformer layers* in the middle, and a *classifier* (CLS) at the top. Green blocks represent parameters available from fine-tuning, and yellow blocks represent parameters that are initialized and optimized after fine-tuning. (a): pruning removes unimportant parts of the original model and rewires the connection; (b): early exiting adds extra classifiers for intermediate transformer layers; (c): Distillation initialized a new student model and distill knowledge from the original teacher model. 16

3.1 DeeBERT model overview. There are n layers in total; blue blocks are transformer layers and orange blocks are classifiers. 23

3.2 DeeBERT quality and efficiency trade-offs for $\text{BERT}_{\text{BASE}}$ and $\text{RoBERTa}_{\text{BASE}}$ models. 28

3.3 Comparison between expected saving (x-axis) and actual measured saving (y-axis), using $\text{BERT}_{\text{BASE}}$ and $\text{RoBERTa}_{\text{BASE}}$ models. 29

3.4 Accuracy of each classifier for $\text{BERT}_{\text{BASE}}$ and $\text{RoBERTa}_{\text{BASE}}$. 30

3.5 Results for $\text{BERT}_{\text{LARGE}}$ and $\text{RoBERTa}_{\text{LARGE}}$. 31

3.6 Number of output examples by layer for $\text{BERT}_{\text{BASE}}$ and $\text{RoBERTa}_{\text{BASE}}$. Each plot represents a separate entropy threshold S. 32

4.1 Multi-output structure of early exiting BERT. There are n layers in total; blue blocks are transformer layers, orange blocks are classifiers, and green blocks are exiting decision making modules. The green blocks are the difference between this diagram and Figure 3.1. 34

4.2 Results for $\text{BERT}_{\text{BASE}}$. 42

4.3 Results for $\text{RoBERTa}_{\text{BASE}}$ (top row) and $\text{ALBERT}_{\text{BASE}}$ (bottom row). 43

4.4 Comparing layer-wise score of *Alternating* with LTE-based early exiting on top of *Alternating*. $\text{BERT}_{\text{BASE}}$ is the backbone. 46

4.5 Comparison of each layer's score and learned certainty. Yellow box-plots: distribution of certainty at each layer; blue curve: relative score. 48

4.6 Comparison between BLEU–confidence correlation (red and blue) and layer-wise scores (black). 49

5.1 Overview of early exiting BERT for document ranking. Blue blocks are transformer layers and orange blocks are classifiers. The model architecture is the same as Figure 3.1. 52

5.2 Comparison between layer-wise scores and early exiting trade-offs. 57

5.3 Comparison between different values of S_p (curves) and S_n (points on a curve). . 58

5.4 Comparison between different values of S_p (curves) and S_n (points on a curve). This is similar to Figure 5.3, but using a dataset with a much higher relevant document proportion. 59

6.1 Example of a selective classifier that makes a prediction for a confident example (left) and abstains for an uncertain one (right). 62

6.2 Risk–coverage curves of $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$ models with SR and MC confidence estimators. The legend applies to all sub-plots. 65

6.3 Selective prediction performance of MC-dropout with different numbers of repetitive runs (x-axis) and dropout rates (marked in the legend). $\text{BERT}_{\text{BASE}}$ is used here. The legend applies to all sub-plots. 70

6.4 Accuracy–efficiency trade-offs by using classifier cascades. All examples are first evaluated by LSTM, and then we compare three ways of choosing examples to send to the more sophisticated model ($\text{BERT}_{\text{BASE}}$): random selection (Random), SR without regularization (SR), and SR with history error regularization (SR-hist.). The legend applies to all sub-plots. 75

7.1 Different orderings of the same set of operators have similar trade-off curves. The title of each subfigure shows the set of operators; each color represents a dataset; each marker shape represents a component ordering. 83

7.2 Comparing the results of a single run (solid lines; same as the ones from Figure 7.1) and the results from multiple runs (dashed lines for the mean and shaded areas for 95% confidence intervals). 84

7.3 Estimating the trade-off curves of target pipelines based on the results of individually applying operators. Green curves: measured trade-off curves of target pipelines; blue curves: measured trade-off curves of individually applying the operator **E**; orange curves: estimated trade-off curves for the target pipelines. . . . 87

8.1 Comparison between the *limit* scores of $\text{BERT}_{\text{BASE}}$ (brown) and $\text{BERT}_{\text{LARGE}}$ (blue). The y-axis shows the score (F1 for MRPC and accuracy for QNLI) relative to vanilla fine-tuning of $\text{BERT}_{\text{LARGE}}$. The red arrow shows the difference between the two models when they both use the same number of layers (12). 93

Chapter 1

Introduction

In the natural language processing (NLP) and information retrieval (IR) communities, the debate of *smarter* vs *bigger* (better model design and execution vs more data and resources) has been going on forever.[2] In recent years, driven by the notable success of pre-training methods such as BERT (Devlin et al., 2019), the *bigger* team is starting to dominate the field. The idea can be concisely summarized as "重剑无锋 大巧不工" (the heaviest sword does not have a sharp edge, and the greatest ingenuity is never complicated).[3] The heavy sword here is large-scale pre-trained language models, and they are heavy in two ways. First, they require a large amount of training data: from pre-trained word embeddings (Mikolov et al., 2013; Pennington et al., 2014) to pre-trained contextualized representations with Long Short-Term Memories (LSTMs) (Peters et al., 2018), datasets of billions of words are necessary.[4] Second, large and sophisticated model architectures, namely the transformer model (Vaswani et al., 2017), have become the *de facto* standard choice (Radford et al., 2019; Devlin et al., 2019). Since the advent of BERT (Devlin et al., 2019), different types of pre-trained language models have started to emerge, such as XLNet (Yang et al., 2019), RoBERTa (Liu et al., 2019), T5 (Raffel et al., 2020), Electra (Clark

[2]One of the latest: https://twitter.com/lintool/status/1377661127283326977.

[3] This is a quote from the kungfu novel *The Return of the Condor Heroes* by the Chinese writer *Jin Yong*. The original context is that for fighters with enough strength, they can simply use heavy swords and brute force, while sharp sword edges or complicated kungfu skills are not necessary.

[4] The size of the latest dataset, C4, on which the T5 model (Raffel et al., 2020) is trained, is 800 GB after cleaning and 6 TB before: https://www.tensorflow.org/datasets/catalog/c4.

et al., 2020), GPT-3 (Brown et al., 2020), Switch Transformer (Fedus et al., 2021), and so on. Despite the increasingly large sizes and fancy model training techniques, the core idea behind these models remains the same: get as much data as possible, find a large model with enough capacity to learn the pattern of data, and burn lots of energy to power GPUs and even TPUs to train the model.

This simple idea works well, and these increasingly large pre-trained language models have brought significant improvements to both academia benchmarking tasks and industry applications. For example, on the General Language Understanding Evaluation benchmark (GLUE) (Wang et al., 2018a), which is one of the most widely-used benchmarks covering a wide range of common NLP applications, the average score has improved from the pre-BERT state of the art's 74, to BERT's 82, and to T5's 90;[5] commercial search engines such as Google[6] and Microsoft Bing[7] are also applying BERT-like models to improve the quality of search results.

1.1 Challenges

All is fine, except the sword is getting too heavy in a lot of cases, where the large amount of hardware resource and computing time is a luxury. Despite their exciting power, these transformer-based models are notorious for being enormous in size. For example, the number of model parameters has increased from a two-layer bi-directional LSTM's 1 million, to $\text{BERT}_{\text{LARGE}}$'s 300 million, to GPT3's 175 billion, and to Switch Transformer's over 1 trillion. Moreover, these new models are slow in both training and inference. For example, on MRPC (Dolan and Brockett, 2005), one of the datasets from GLUE, $\text{BERT}_{\text{BASE}}$'s inference for one example takes about 100 milliseconds with a high-end GPU (Xin et al., 2020a), and the number of FLOPs is several orders of magnitude higher than that of an LSTM (Xin et al., 2021b). When the number of inference examples increases, or when these large models need to be deployed on low-end or edge devices, the cost of running the models and inference latency become non-trivial factors. Therefore,

[5] https://gluebenchmark.com/leaderboard

[6] https://blog.google/products/search/search-language-understanding-bert

[7] https://azure.microsoft.com/en-us/blog/bing-delivers-its-largest-improvement-in-search-experience-using-azure-gpus/

improving the efficiency of transformer-based NLP models has become an important and urgent problem.

In this thesis, we discuss model efficiency for NLP tasks in the transformer era. We hope that *smarter* can be combined with *bigger*—design better efficiency methods for large models and make the heavy sword accessible to more kungfu fighters that do not possess the originally necessary brute force. The application of transformers typically involves three stages: pre-training, fine-tuning, and inference. Among the three stages, we choose to focus on inference efficiency for the following reasons: Pre-training is typically done with abundant resources and model checkpoints are often publicly shared for repetitive use. Fine-tuning is done on relatively small task-specific datasets without strict time constraints. Inference, by contrast, needs to be performed at a much higher frequency and with a much lower tolerance for latency. Therefore, improving inference efficiency is broadly meaningful to the community.

1.2 Early Exiting and Beyond

The transformer model typically consists of several transformer layers, all of which share the same model architecture. Therefore, a natural idea for efficient inference is to reduce the *depth* of the model by using only a fraction of all layers. More concretely, the *early exiting* idea is an adaptive computation method that dynamically executes each layer sequentially and terminates inference when a certain layer's output satisfies the exiting criterion (Figure 1.1). The inspiration comes from a well-known observation in the computer vision community: in deep convolutional neural networks, deeper layers typically produce more detailed and finer-grained features (Zeiler and Fergus, 2014). For data points where coarse-grained features produced by shallower layers suffice, the early exiting idea reduces the computation for deeper layers and improves inference efficiency. We make the first attempt of applying the early exiting idea to NLP tasks on sequence classification tasks from the GLUE benchmark, including tasks such as sentiment analysis (Socher et al., 2013), paraphrasing (Dolan and Brockett, 2005), entailment classification (Williams et al., 2018), linguistic acceptability judgment (Warstadt et al., 2019), etc. In Chapter 3, we provide detailed design and experiments, using transformer models BERT (Devlin et al., 2019) and its

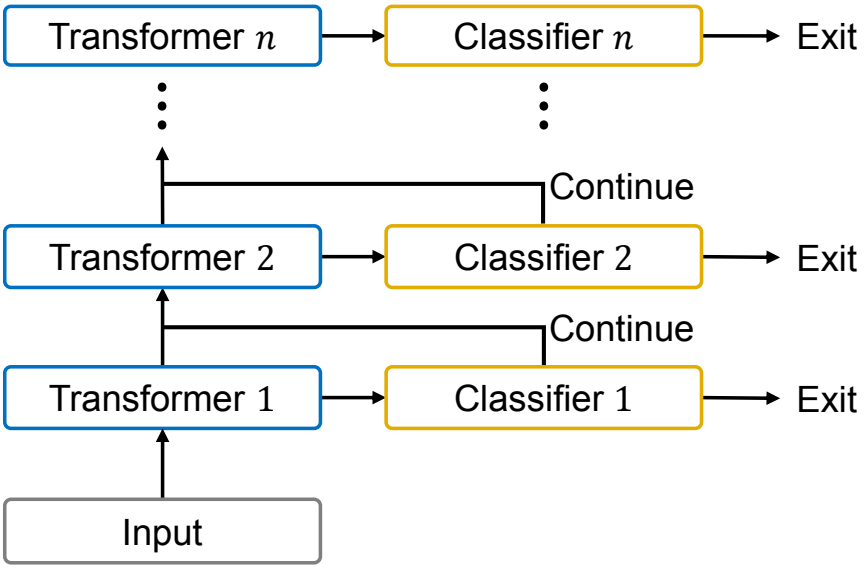


Figure 1.1: Early exiting model overview. There are n layers in total; blue blocks are transformer layers and orange blocks are classifiers. This diagram will be used in Chapter 3 and Chapter 5.

variant RoBERTa (Liu et al., 2019) as the *backbone models*.[8] We use the most straightforward way to enable early exiting: take a fine-tuned transformer model, freeze its parameters, add extra classifiers to it, and only fine-tune these additional classifiers. Since we work with classification tasks, the output of each classifier is a probability distribution, from which we can easily obtain the confidence of the prediction (simply take the maximum entry of the distribution). We then compare the confidence with a threshold, based on which we make the early exiting decision. We show that the idea is capable of reducing up to 50% of inference time, depending on the dataset and backbone model. We publish the discoveries as the DeeBERT paper (Xin et al., 2020a), which is the first paper to discuss early exiting for transformers.

The combination of early exiting and transformers is not limited to any specific task, dataset, or backbone model. Rather, it can be extended to other scenarios, with adequate modifications. We discuss several such extensions in Chapters 4 to 5.

The first extension is from classification to regression, described in the first part of Chapter 4. For regression tasks, since the output of the model is just a single value, it is hard to estimate how

[8]Backbone model is the model on which we apply the efficiency method.

confident the model is in this output. We train an auxiliary module alongside the main model to explicitly estimate the model's confidence. During training, the auxiliary module predicts the mean squared error (MSE) of the main model; during inference, the output of the auxiliary module replaces the confidence of distribution and is compared with the threshold. We publish the work as the first part of the BERxiT paper (Xin et al., 2021a).

The second extension is for low-resource datasets, described in the second part of Chapter 4. The original method to fine-tune the model is intuitive, but may not be ideal, especially in the case where training data is not enough to train the entire model well. Another straightforward way is to start from a pre-trained model and fine-tune the entire multi-output structure together. This fine-tuning method sacrifices the accuracy of the final layer but improves the intermediate layers. We discuss a combination of the two fine-tuning methods mentioned above, which leads to a better balance between all layers and provides better accuracy–efficiency trade-offs for low-resource datasets. We also provide analyses that reveal interesting properties of the combination of early exiting and transformers. We publish the work as the second part of the BERxiT paper (Xin et al., 2021a).

The third extension is for document reranking, described in Chapter 5. Document reranking is part of the *retrieve–process pipeline*, also known as the retriever–reader system (Chen et al., 2017). The pipeline is useful for tasks such as web search (Nogueira and Cho, 2019), grounded generation (Lewis et al., 2020b), open domain question answering (Karpukhin et al., 2020), etc. In this pipeline, the first stage is to use a retriever to retrieve information needed for the task, and the second stage is to process retrieved information to generate the final output. The first stage retriever is typically a token-based model such as BM25 (Robertson and Jones, 1976), which is fast but relatively inaccurate. The second stage processor (or reader) is typically a transformer model, which is more accurate than the first stage retriever but also slower. In our experiments, we focus on document reranking for web search. It is originally formulated as a binary classification task: the query–document pair is classified as relevant or irrelevant. However, considering the two classes are highly asymmetric, naïvely applying the vanilla early exiting leads to poor performance. We specifically design an asymmetric early exiting method, which greatly improves over the original baseline. Our analyses on choosing asymmetric early exiting thresholds also show the necessity for such a method. We publish the work as the Early Exiting MonoBERT (EEMB) paper (Xin et al., 2020b).

Early exiting is the idea of adaptively allocating computing resources to data points to match their "difficulty". Extrapolating this idea, we can allocate *no* computing resource (or as little as possible) to data points that are the most difficult and hopeless to get right. This is the selective prediction scenario (Geifman and El-Yaniv, 2017) and we discuss its combination with transformers in Chapter 6. We present a simple regularization method to improve confidence estimation for transformers and we demonstrate with experiments that the regularization indeed produces transformer models with confidence that is more aligned with their correctness. We further discuss selective prediction and confidence estimation in two applications.

- The *no-answer* problem: in a classification task, if a test set example has a label that is unseen in the training set, the model should abstain from making predictions at all. We enable this by thresholding the model's confidence. The proposed regularization trick improves the model's accuracy under this setting.
- The classifier cascade: we have a series of models of various accuracy and efficiency, and we try to allocate different inference examples to them to achieve the best accuracy–efficiency trade-off. We show that making the allocation according to the confidence of the base model (faster but less accurate) provides improvements compared with the random allocation baseline. Applying the regularization further improves the trade-off.

We publish the work as the Art of Abstention paper (Xin et al., 2021b).

Early exiting is just one of the efficiency methods, and there is nothing preventing us from combining it with others. In Chapter 7, we discuss the combination of early exiting and other efficiency methods, such as pruning (McCarley et al., 2019), distillation (Sanh et al., 2019), and quantization (Shen et al., 2020). Conceptually, we consider each efficiency method as an *operator*, apply them sequentially to a pre-trained transformer model, and study the accuracy–efficiency trade-off of the final model. We study the commutativity and cumulativeness of these operators. For commutativity, we demonstrate that the operators are practically commutative: swapping the order of applying them has little effect on the final result. For cumulativeness, we show that, with minor exceptions, these operators are cumulative: the accuracy and efficiency of the final model can be estimated by combining the accuracy and efficiency of individual operators. From the empirical perspective, these observations make it easier to apply multiple efficiency operators to

transformer models and to estimate their performance without repetitive experiments. From the theoretical perspective, it also provides an interesting topic for studying the interaction between different efficiency methods and evaluating their contribution using methods such as Shapley value (Shapley, 1997). We summarize the work as the Efficiency Pipeline manuscript (Xin et al., 2022).

Finally, in Chapter 8, we conclude the thesis and discuss interesting future directions for further research on this topic.

1.3 Contributions

There are six major contributions in the thesis, arranged in Chapters 3 to 7. We summarize these contributions here:

- Inspired by the fact that different layers of the transformer model extract features of different granularity, we initialize the exploration to accelerate transformer models with early exiting. In addition to necessary modifications of the model architecture, we also develop a two-stage fine-tuning strategy that preserves the original BERT model's effectiveness. We conduct experiments to show that the early exiting idea is able to save up to 50% of inference time for sequence classification tasks. Furthermore, we analyze the early exiting pattern and show that for each example, the model is able to adaptively choose the exiting layer with the best accuracy–efficiency trade-off.

- Following the success of applying early exiting to sequence classification tasks, we design the learning-to-exit idea to extend early exiting to sequence regression. We add a one-layer feed forward network (FFN) as an auxiliary confidence estimator to explicitly estimate the confidence of the model, which is not directly available in regression tasks. Our experiments show that the learning-to-exit idea is able to capture model confidence effectively and thereby speedups regression tasks by more than $2\times$.

- Different from straightforward fine-tuning methods used in other baselines, we present a novel alternating fine-tuning method that is specifically suitable for low-resource datasets.

By combining training objectives from others, alternating fine-tuning is able to find the intersection between optimal regions for different layers in the parameter space. The benefit of the novel fine-tuning method is greater for low-resource datasets where balancing different layers is more difficult. Our experiments show that for low-resource datasets, the accuracy of the model produced by alternating fine-tuning is $2 \sim 3\%$ higher than that from joint fine-tuning, given the same efficiency constraint.

- Document reranking is a very important and unique task for NLP and IR. We analyze the intrinsic asymmetry of this task and propose asymmetric early exiting for it. We conduct experiments to show that we can achieve more than $3\times$ acceleration with less than 1% drop in model effectiveness. We further analyze the behavior of different early exiting thresholds, which further expose the asymmetry of document reranking.

- We explore the idea of selective prediction for transformers, and propose a regularization method to improve the accuracy of confidence estimation. We evaluate and compare the effectiveness of different methods of confidence estimation in not only the standard setting, but also two other virtual settings that we propose, namely the no-answer problem and the classifier cascade.

- We present a novel conceptual framework to consider multiple efficiency methods as operators, and conduct extensive experiments to study their properties. We show their commutativity and cumulativeness, which are highly important and useful for designing, evaluating, and applying efficiency operators in practice. Furthermore, the framework provides a new perspective for understanding them theoretically.

Chapter 2

Background

The advent of BERT (Devlin et al., 2019) in late 2018 changes the community with its great power, quickly rendering pre-trained transformers (Vaswani et al., 2017) as the *de facto* standard approach for many NLP research topics. In this chapter, we introduce the transformer model and summarize previous efficiency methods developed for transformers.

2.1 Model Architecture: Transformers and Training

2.1.1 Transformers Architecture

The transformer architecture (Vaswani et al., 2017) is proposed to supersede Recurrent Neural Networks (RNN) (Pineda, 1987), and more specifically, its popular variants Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Units (GRU) (Cho et al., 2014), for processing sequential inputs. Similar to RNNs, transformers take as input a vector sequence $\{\mathbf{x}_t\}_{t=1}^n$ and output another vector sequence of the same length $\{\mathbf{y}_t\}_{t=1}^n$. For sequence processing, one of the most important challenges is to model the *interaction* between tokens. In RNNs, input tokens interact with each other through the recurrent function and decide the output for the next recurrent step. The transformer architecture, on the other hand, completely relies on the *attention mechanism* (Bahdanau et al., 2014) for token interaction. Not having to

Figure 2.1: The model structure of a transformer layer (left) and the multi-head attention mechanism (right).[9]

rely on recursion enables transformers to run in a highly parallel manner and eventually allows them to grow in size, capacity, and power.

Attention is a method to give a "summary" of a set of inputs based on a query, and an input contributes more to the summary if the input is more relevant to the query. Concretely, the summary is the weighted sum computed as follows:

$$\mathbf{y}_t = \sum_{i=1}^{n} a_{t,i}\mathbf{x}_i. \tag{2.1}$$

Here, $a_{t,i}$ are referred to as *weights* and $\mathbf{x}_i$ as *value* vectors (the inputs mentioned above). Despite its fancy name, attention is simply a way to compute the weights $a_{t,i}$. It takes as input a *query* vector $\mathbf{q}$ and a set of *key* vectors $\{\mathbf{k}_i\}_{i=1}^n$, and outputs n weights corresponding to each of the key vectors:

$$\text{Attn}(\mathbf{q}; \mathbf{k}_1, \cdots, \mathbf{k}_n) = \text{softmax}(\mathbf{q}^\top \mathbf{k}_1, \cdots, \mathbf{q}^\top \mathbf{k}_n) \in [0,1]^n. \tag{2.2}$$

In the case of transformers, $a_{t,i}$ in Equation (2.1) is the ith entry of $\text{Attn}(\mathbf{x}_t; \mathbf{x}_1, \cdots, \mathbf{x}_n)$. In fact, in Equations (2.1) to (2.2), the *query*, *key*, and *value* vectors are all $\{\mathbf{x}_t\}_{t=1}^n$, just in different arrangements of the subscripts. In practice, a more sophisticated version of attention, namely *multi-head* attention, is used for transformers. In each *head* of multi-head attention (marked with a superscript k), the input vectors $\{\mathbf{x}_t\}_{t=1}^n$ are mapped to separate *query*, *key*, and *value* spaces

[9] These two diagrams are taken from the original Transformer paper (Vaswani et al., 2017).

Hyper-parameter	Typical value	Explanation
Layers	12	Number of transformer layers used
Hidden size	784	Dimension of vectors $\mathbf{x}_t$
Attention heads	12	The max attention head index k
Max sequence length	512	The max time step t

Table 2.1: Hyper-parameters used in a typical transformer ($\text{BERT}_{\text{BASE}}$).

with the projection matrices $W_Q^{(k)}$, $W_K^{(k)}$, and $W_V^{(k)}$. Finally, the output of multi-head attention is the concatenation of outputs of all individual heads, followed by an output projection matrix W_O:

$$\mathbf{y}_t = W_O \operatorname*{Concat}_k \left[\sum_{i=1}^{n} a_{t,i}^{(k)} W_V^{(k)} \mathbf{x}_i \right], \tag{2.3}$$

$$\{a_{t,i}^{(k)}\}_{i=1}^{n} = \text{Attn}(W_Q^{(k)}\mathbf{x}_t; W_K^{(k)}\mathbf{x}_1, \cdots, W_K^{(k)}\mathbf{x}_n). \tag{2.4}$$

Here $\operatorname*{Concat}_k$ stands for concatenating values from different attention head indices k.

The multi-head attention mechanism deals with input token interaction. Apart from the attention sublayer, the transformer layer has another FFN sublayer (Figure 2.1). For each sublayer, there is a residual connection (He et al., 2016) that goes around it, and a layer normalization module (Ba et al., 2016) that follows the sublayer and the residual connection. All these combined constitute one transformer layer, and in most applications, a transformer *model* is a stack of multiple transformer *layers*. Typically, all the layers share the same architecture, but they have independent weights.

It may be useful to list hyper-parameters used in transformers (Table 2.1), since many methods to improve transformers' efficiency boil down to decreasing one or more hyper-parameters.

2.1.2 Pre-training, Fine-tuning, and Inference for Transformers

We now introduce how to train a transformer model in what is known as the *pre-train and fine-tune* pipeline. In the original paper (Vaswani et al., 2017), transformers are merely proposed as

a model architecture and deployed in a standard way: randomly initialize all parameters, train the model on the training set, and evaluate the model on the test set. Inspired by work on *pre-training* RNNs on unlabelled data before *fine-tuning* it on the target training set (Dai and Le, 2015; Peters et al., 2018), several papers apply the pre-training trick to transformer-based models, including GPT (Radford et al., 2019), BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), XLNET (Yang et al., 2019), ALBERT (Lan et al., 2020), ELECTRA (Clark et al., 2020), etc.

At the very beginning, all parameters in the model are randomly initialized. *Pre-training* aims to update model parameters on unsupervised corpora. The high-level idea of the pre-training task is to predict a part of the sequence given the context information, and in order to do well, the model needs to have the ability to model textual data. There are two categories of unsupervised pre-training objectives: (1) Autoregressive—the model tries to predict the token x_t given the preceding subsequence $\{x_i\}_{i=1}^{t-1}$; (2) Autoencoding—the model is given a sequence $\{x_i\}_{i=1}^{n}$ where one or more tokens are masked, and the model tries to predict these masked tokens given the unmasked ones. BERT, for example, uses an autoencoding objective, masked language modeling, where random words are replaced with a mask and the model learns to recover the masked word based on its context. Regardless of the concrete objective, after pre-training, the model can be used as a text encoder to produce a sequence of output vectors where information of the input token sequence is encoded. Given a downstream task, the model is further *fine-tuned* by feeding the output vectors of the encoder to a newly-added task-specific classifier to make predictions. For example, for sequence-level classification tasks such as sentiment analysis, one of the output vectors is fed into an FFN that serves as the sentiment classifier; for token-level classification tasks such as span prediction and named entity recognition, each of the output vectors is fed into an FFN that serves as the token type classifier. Parameters from both the original encoder and the downstream-task-specific classifier are jointly updated during fine-tuning. The entire model is finally used for *inference* and *evaluation*, in which case, transformer models are used in the same way as traditional machine learning models.

Transformer models are designed to accommodate a sequence pair (s_1, s_2) as input, and the pair is concatenated as follows:

$$\texttt{[CLS]}\ s_1\ \texttt{[SEP]}\ s_2\ \texttt{[SEP]}$$

Here, `[CLS]` and `[SEP]` are special tokens for marking the beginning of input and separating

the two sequences.[10] In the case of using a single sequence as input, s_2 is simply an empty string. After concatenation, the input sequence is tokenized with WordPiece embeddings (Wu et al., 2016), where special tokens such as `[CLS]` and `[SEP]` are also considered as a part of the vocabulary and have their own embeddings. An embedding layer, which is randomly initialized before pre-trained, converts the input token sequence into a sequence of embeddings, and feeds them into the first layer of the transformer. The output corresponding to the `[CLS]` token is usually regarded as the "sequence" embedding and is used for classification for sequence-level classification tasks.

Transformers are much more complex than RNNs, not only conceptually but also computationally. Therefore, improving the efficiency of transformer-based models has become a more interesting and important problem as they become dominating in NLP applications. Efforts to improve efficiency in the three stages, *pre-training, fine-tuning, and inference*, have mainly been put on pre-training and inference. Working in an unsupervised manner, pre-training requires only plain text corpora, which can be extremely large in size. This makes pre-training extremely computationally extensive and time consuming.[11] Luckily, for people who do not work directly on pre-training, pre-trained models are often publicly available and can be directly used. Fine-tuning, on the other hand, is typically done on a labeled dataset that is inevitably smaller. Efficiency for fine-tuning is rarely a concern for most scenarios.[12] Comparatively, people care more about inference efficiency, especially for online cases where users are sensitive to latency.[13]

2.2 Related Work: Efficiency Methods for Transformers

We introduce existing efficiency methods for transformers applied to NLP tasks. We do not aim to be exhaustive here, but instead focus on work that is more related to subsequent chapters.

[10] CLS is short for classifier and SEP for separator.

[11] Pre-training time: $\text{BERT}_{\text{BASE}}$ takes 4 days on 16 TPUs; $\text{RoBERTa}_{\text{LARGE}}$ takes 1 day on 1024 GPUs; the notoriously enormous GPT-3 (Brown et al., 2020) takes more than 300 GPU-years.

[12] Fine-tuning $\text{BERT}_{\text{BASE}}$ on the SST-2 sentiment analysis dataset takes about an hour on one GPU.

[13] In a certain IR case, the latency per query is 2000ms for BERT and less than 20ms for most pre-BERT models (Hofstätter and Hanbury, 2019).

2.2.1 Structured Pruning

Pruning (Han et al., 2015) aims to reduce model size and/or accelerate model execution by removing unnecessary or unimportant parts of the model. Conventionally, effort on pruning focuses on *unstructured* pruning in that it removes individual weights of parameter matrices and vectors, e.g., the work by Frankle and Carbin (2019). While unstructured pruning creates sparse weights in the model, it does not directly improve runtime efficiency, since the sparsity in weights does not easily convert to a reduction in computation. *Structured* pruning (Anwar et al., 2017), on the other hand, removes model parts at a higher level (e.g., an entire row or column of the weight matrix), and can be directly applied to reduce computation (Figure 2.2a). In the case of transformers, structured pruning typically focuses on reducing the number of layers, the length of input/intermediate sequences, the number of attention heads, etc. After structured pruning, the model typically requires *rewiring*, where we reorganize the remaining structure to form a compact smaller model.

Number of layers In the computer vision community, there has been work on using reinforcement learning to learn a policy to skip some of the modules/layers of a neural network (Lin et al., 2017; Wang et al., 2018b; Wu et al., 2018). For NLP, Layerdrop (Fan et al., 2020) proposes to reduce the number of layers by using structured dropout. Different from traditional dropout, where single weights are randomly disconnected from the network during training, structured dropout randomly disconnects larger units such as an entire layer. Conceptually, structured dropout is to traditional dropout as structured pruning is to unstructured pruning. By dropping out layers during fine-tuning, Layerdrop makes the model less susceptible to accuracy degradation brought by layer removal during inference.

Early exiting is another important idea for reducing the number of layers, but since it is closely related to this thesis, we will discuss the details in Section 2.2.2.

Input/intermediate sequence length In the original setting for transformer models, each layer's input and output sequences have the same length. However, an intermediate transformer layer can also take only a part of the previous layer's output as the input for its multi-head attention computation, thereby reducing the effective intermediate sequence length. In fact, most pre-trained

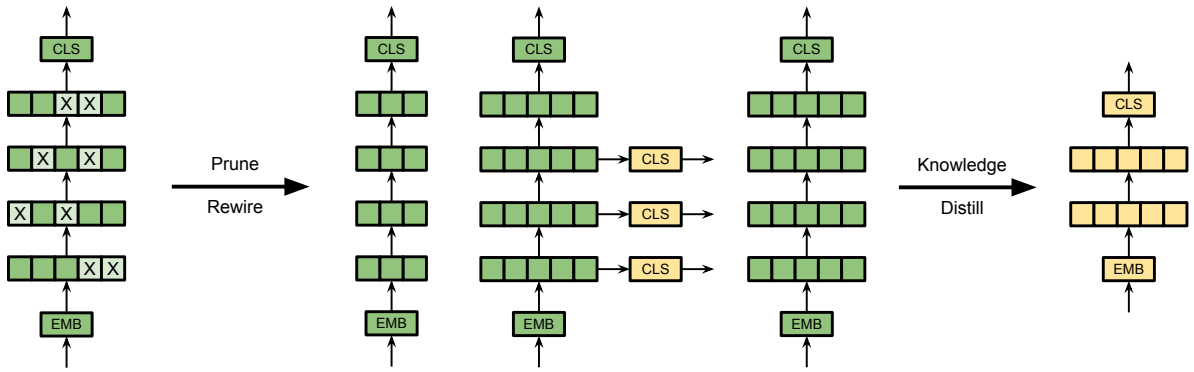
transformer models use a fixed sequence length that is much larger than needed by downstream fine-tuning tasks,[14] and this is a heavy waste of computation (recall that the complexity of the attention mechanism is quadratically proportional to sequence length). PoWER-BERT (Goyal et al., 2020) is the first attempt to do so at the fine-tuning stage: it gradually reduces the sequence length as going into deep layers, before eventually reducing the sequence length to 1 at the final layer, which is enough for sequence-level prediction. Length-Adaptive Transformer (Kim and Cho, 2020) extends the idea to tasks other than sequence-level prediction (such as token-level prediction), by first reducing the sequence length and then recovering missing tokens' outputs. Funnel Transformer (Dai et al., 2020) further extends the idea to pre-training: at each transformer layer, the sequence is shortened from the previous layer by reducing the number of query vectors in the multi-head attention mechanism; at the final layer, the sequence length is recovered by merging outputs of previous layers.

Number of attention heads Following the effort of Voita et al. (2019) to analyze redundancy in attention heads, Michel et al. (2019) show that directly removing attention heads *after* training/fine-tuning does not significantly degrade the model's effectiveness and argue that in a lot of cases, multiple attention heads may not be necessary. MobileBERT (Sun et al., 2020) leverages the observation to propose a faster BERT model: at each transformer layer, a funnel-like structure is used to first reduce the model's hidden size (the dimension of vectors to be processed by the multi-head attention mechanism, which is the most time-consuming part), and then to recover the original hidden size at the end of the layer. In addition to the smaller hidden size, fewer attention heads are used. Experiments show that MobileBERT achieves comparable accuracy with BERT while using fewer parameters and less inference time.

2.2.2 Early Exiting

Early exiting is an important structured pruning idea to reduce the number of layers: it terminates the computation of a model at an intermediate layer and immediately makes the final prediction without going through the remaining layers (Cambazoglu et al., 2010). There have been a number

[14] For example, BERT is pre-trained with sequence length 512, while the maximum sequence length required by GLUE tasks is 128.



(a) Pruning (b) Early exiting (c) Distillation

Figure 2.2: Diagrams for efficiency methods. The model consists of an *embedding layer* (EMB) at the bottom, a few *transformer layers* in the middle, and a *classifier* (CLS) at the top. Green blocks represent parameters available from fine-tuning, and yellow blocks represent parameters that are initialized and optimized after fine-tuning. (a): pruning removes unimportant parts of the original model and rewires the connection; (b): early exiting adds extra classifiers for intermediate transformer layers; (c): Distillation initialized a new student model and distill knowledge from the original teacher model.

of publications on early exiting for transformer models, and they roughly fall into two categories which we dub as *explicit* early exiting and *implicit* early exiting. The difference between these two categories lies in how the model makes the decision of continuing/exiting after each layer. In explicit early exiting, a separate module of the model, called the *halting unit*, explicitly predicts whether or not to halt the computation; while in implicit early exiting, the decision is made based on the model's prediction, e.g., comparing the maximum predicted probability with a threshold.

Explicit early exiting The idea of explicit early exiting starts from Adaptive Computation Time (ACT) (Graves, 2016a). A halting unit is injected into the model itself, and thus the model makes two predictions together: the *task prediction*, e.g., the probability distribution of a classification task, and the *exiting prediction*, i.e., whether to exit here and to output the latest task prediction. It is assumed that the task prediction gets better and better as more computation resources are used, and therefore the latest prediction is used. During training, the loss function

is computed from the chosen task prediction; a layer penalty is also applied as a regularizer to encourage the model to use fewer layers while maintaining the prediction quality. Universal Transformer (UT) (Dehghani et al., 2019) is proposed as the ACT application on transformers. In typical transformer models such as BERT, each word in the input sequence is processed by a fixed number of layers (with possibly different parameters), while in UT, each word is processed by the same transformer layer (i.e., transformers layers with shared parameters), but for a different number of times, depending on the halting unit's output. Following UT, Depth-Adaptive Transformer (DAT) (Elbayad et al., 2020) and Faster-Depth-Adaptive transformer (FDAT) (Liu et al., 2021) extend the idea of UT in the following two ways: (1) transformer layers no longer share parameters; (2) instead of training the halting unit along with the "main" model, it is now trained with explicit supervision—in DAT, the "ground truth" layer is the one with the highest accuracy/likelihood (depending on the task), and in FDAT the "ground truth" layer is chosen by estimating mutual information between each word and the label or estimating the reconstruction loss when each word is masked from the language model.

Implicit early exiting The idea of implicit early exiting for deep learning model is proposed in BranchyNet (Teerapittayanon et al., 2016). In implicit early exiting, the model makes only the task prediction, and the exiting decision is implicitly made based on properties of the task prediction. Our published work, DeeBERT (Chapter 3; Xin et al., 2020a), falls into this category (Figure 2.2b). Somewhat coincidentally, Right-Tool-for-the-Job (RTJ[15]) (Schwartz et al., 2020) and FastBERT (Liu et al., 2020), which also apply implicit early exiting to NLP tasks, are published at the very venue as DeeBERT. The core idea is very similar, and the differences lie in how the model is fine-tuned and how exiting decisions are made. RTJ fine-tunes the entire model (i.e., all transformer layers and classifiers) by minimizing the simple sum of all layers' loss functions, and FastBERT fine-tunes the model with self-distillation, as proposed by Zhang et al. (2019a) and Phuong and Lampert (2019) for computer vision tasks. Both RTJ and FastBERT make exiting decisions by comparing the *confidence* (the maximum predicted probability) with some thresholds. PAtience-Based-Early-Exiting (PABEE) (Zhou et al., 2020) comes out not more than a few months after those three. It trains the network by minimizing a weighted sum of all layers' loss functions (the weight is the layer's index), and it makes the prediction based

[15]Short for *Right Tool for the Job*—the paper does not provide a concise name for the method.

on "patience"—if the model's prediction is stable for a few layers (predicting the same class for classification or predicting close values for regression), then the model is confident enough to halt the computation. Cascade Transformer (Soldaini and Moschitti, 2020) is another instance of implicit early exiting that specifically deals with ranking problems: it works on a batch of examples (likely documents to be ranked), and discarded a fixed proportion of non-relevant examples at each layer, based on their predicted relevance.

Early exiting can be regarded as constructing a series of models where the computation of previous models can be reused. In this sense, it is also closely related to literature from the computer vision community (Huang et al., 2018; Yu et al., 2019).

2.2.3 Other Methods

Knowledge distillation Distillation (Hinton et al., 2015) aims to "distill" knowledge from a large, effective, and likely costly *teacher* model to a small, efficient, and likely less effective *student* model (Figure 2.2c). The distillation process is, typically, first using the teacher model to annotate a dataset and then training the student model with the annotation. Differences between each method lie in how to annotate with the teacher, how to choose the student architecture, and how to train the student with the teacher's annotation. Tang et al. (2019) perform task-specific distillation into non-transformer architectures such as LSTMs: the teacher is a BERT model that has already been fine-tuned on a specific task; a transfer dataset is constructed by augmenting the original training set with rule-based approaches; the teacher model is used to annotate the transfer dataset; the student model is trained on the transfer dataset by aligning its prediction logits with the teacher. Patient Knowledge Distillation (PKD) (Sun et al., 2019) also performs task-specific distillation, but the student models are transformers with fewer layers and smaller hidden sizes; furthermore, they perform *patient distillation*, in that they align not only logits predicted by teachers and students, but also hidden states in intermediate layers. DistilBERT (Sanh et al., 2019) extends the idea to both task-agnostic and task-specific distillation: first the student model learns from a pre-trained but not fine-tuned teacher, then it further learns from another teacher that has been fine-tuned on a specific task; the student can also, after the first stage learning, be directly fine-tuned on a specific task instead of learning from a proxy teacher. TinyBERT (Jiao et al.,

2020) combines the above work: the student models are smaller transformers; both task-agnostic and task-specific distillation are considered; both general distillation (aligning prediction logits) and transformer-specific distillation (aligning embedding matrices, attention matrices, hidden states, etc. of intermediate layers) are used.

It is also worth mentioning that knowledge distillation and structured pruning are not mutually exclusive. For example, in some cases (Kim and Cho, 2020), people may first use structure pruning to obtain a smaller model, and then use distillation to train a specific part of the smaller model.

Quantization and hardware-related approaches Quantization (Courbariaux et al., 2014; Lin et al., 2016) aims to use fewer bits than normal to store parameters and perform the computation of a deep learning model, and has been applied to transformer models (Shen et al., 2020; Zhang et al., 2020). Hardware-Aware Transformer (Wang et al., 2020) is proposed to search for an adequate transformer model configuration (e.g., attention head numbers, layer numbers, inter-layer connection styles), based on the given hardware and latency constraints. However, these methods are less related to the thesis and we therefore omit the details.

2.3 Experimental Setup: Datasets, Implementation, and Pre-trained Checkpoints

In this section, we introduce factors in our experimental setup that are available from previous papers and online resources.

Many chapters of the thesis discuss general NLP applications, therefore we use the GLUE benchmark (Wang et al., 2018a). The benchmark includes 10+ datasets, among which we use the following classification and regression ones.

- RTE (Recognizing Textual Entailment) (Bentivogli et al., 2009) is a classification task. Given a pair of sentences, the task is to predict whether the second sentence is an entailment of the first one.

- MRPC (Microsoft Research Paraphrase Corpus) (Dolan and Brockett, 2005) is a classification task. Given a pair of sentences, the task is to predict whether they are semantically equivalent.

- SST-2 (Stanford Sentiment Treebank) (Socher et al., 2013) is a classification task. Given a single sentence of movie review, the task is to predict whether the sentiment is positive or negative.

- QNLI (Question Natural Language Inference) (Rajpurkar et al., 2016; Wang et al., 2018a) is a classification task. Given a question and a sentence, the task is to predict whether the sentence contains the answer to the question.

- QQP (Quora Question Pairs) (Sharma et al., 2019) is a classification task. It is also a semantic equivalence judgment dataset like MRPC, but the sentence pairs are Quora questions.

- MNLI (Multi-genre Natural Language Inference) (Williams et al., 2018) is a classification task. It is also an entailment judgment dataset like RTE, but there are three classes: entailment, contradiction, and neutral.

- STS-B (Semantic Textual Similarity Benchmark) (Cer et al., 2017) is a regression task. Given a pair of sentences, the task is to predict a score from 1 to 5 for how semantically similar the sentences are.

For regression experiments, since GLUE has only one regression dataset, we additionally use SICK (Sentences Involving Compositional Knowledge) (Marelli et al., 2014). Similar to STS-B, given a pair of input sentences, the task is to predict a score from 1 to 5 for how related the sentences are.

Statistics of these classification and regression datasets are listed in Table 2.2.

Our implementation of transformers is adapted from the Huggingface Transformer Library (Wolf et al., 2020). The library also provides pre-trained model checkpoints, and we use the following ones as backbone models in our experiments.

- $\text{BERT}_{\text{BASE}}$-UNCASED and $\text{BERT}_{\text{LARGE}}$-UNCASED

Dataset	# Classes	Train / Dev / Test
RTE	2	2.5k / 0.3k / 3.0k
MRPC	2	3.7k / 0.4k / 1.7k
SST-2	2	67k / 0.9k / 1.8k
QNLI	2	105k / 5.5k / 5.5k
QQP	2	364k / 40k / 391k
MNLI	3	393k / 9.8k / 9.8k
STS-B	1	8.6k / 1.5k / 1.4k
SICK	1	4.4k / 4.9k / –

Table 2.2: Statistics of NLP datasets used in the thesis. STS-B and SICK are regression datasets, thus # Classes = 1.

- RoBERTA$_{\text{BASE}}$ and RoBERTA$_{\text{LARGE}}$
- ALBERT$_{\text{BASE}}$-v2
- DistilBERT$_{\text{BASE}}$-uncased

Chapter 3

Early Exiting for Sequence Classification

As mentioned in Section 2.2.2, early exiting can be regarded as a structured pruning idea to reduce the number of layers in the transformer model. We apply early exiting to pre-trained transformer models such as BERT and RoBERTa, and conduct experiments on sequence classification tasks from the General Language Understanding Evaluation benchmark (GLUE) (Wang et al., 2018a). The results are published in the DeeBERT paper (Xin et al., 2020a).

3.1 Overview

To accelerate inference for BERT-like models,[16] we propose **DeeBERT**: **D**ynamic **e**arly **e**xiting for **BERT**. The inspiration comes from a well-known observation in the computer vision community: in deep convolutional neural networks, higher layers typically produce more detailed and finer-grained features (Zeiler and Fergus, 2014). Therefore, we hypothesize that, for BERT, features provided by the intermediate transformer layers may suffice to classify the easier input examples.

DeeBERT accelerates BERT inference by adding extra classifiers to transformer layers, so that after each transformer layer, there is an exit point through a classifier (Figure 3.1). All transformer layers and classifiers are jointly fine-tuned on a given downstream dataset. At inference time, after an example goes through a transformer layer, it is passed to the adjacent classifier. If the

[16]Henceforth, we often use BERT to refer to other transformer models similar to BERT.

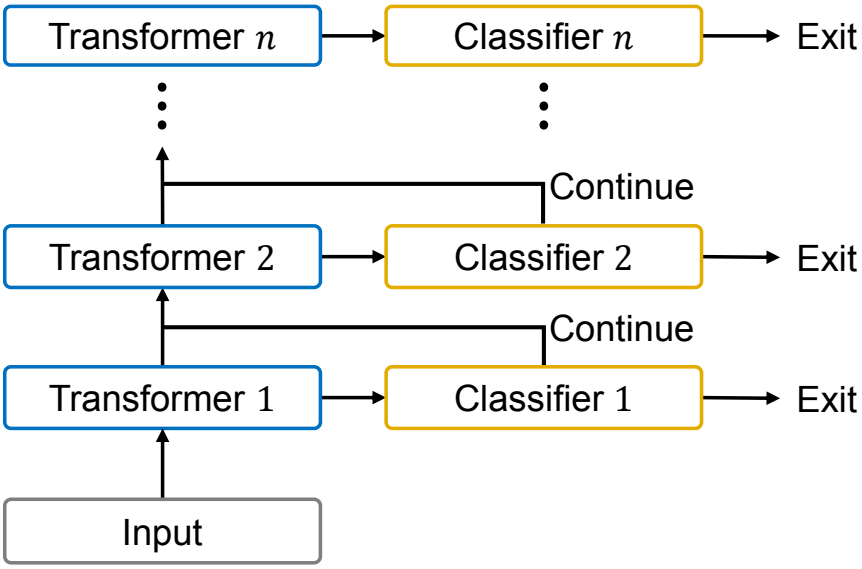


Figure 3.1: DeeBERT model overview. There are n layers in total; blue blocks are transformer layers and orange blocks are classifiers.

classifier is confident of the prediction, the result is returned; otherwise, the example is sent to the next transformer layer.

In this chapter, we conduct experiments on BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) with six sequence classification tasks from the GLUE benchmark, showing that DeeBERT is capable of accelerating model inference by up to ∼40% with minimal model quality degradation on downstream tasks. Further analyses reveal interesting patterns in the models' transformer layers, as well as redundancy in both BERT and RoBERTa.

3.2 Modeling Details

DeeBERT modifies fine-tuning and inference of BERT models, leaving pre-training unchanged. It adds one classifier for each transformer layer. An inference example can exit earlier at an intermediate classifier, without going through the rest of the transformer layers. The last classifier is the classification layer of the original BERT model.

3.2.1 DeeBERT at Fine-Tuning

Backbone model The backbone model is an n-layer pre-trained transformer model. It is a stack of n transformer layers, pre-trained on large corpora in a self-supervised manner. We denote the ith layer's hidden state corresponding to the `[CLS]` token as $\mathbf{h}_i$:

$$\mathbf{h}_i = f_i(x; \theta_1, \cdots, \theta_i), \tag{3.1}$$

where x is the input sequence, θ_i is the parameters of the ith transformer layer, and f_i is the mapping from input to the ith layer hidden state.

Classifiers In the original BERT paper (Devlin et al., 2019), the way to fine-tune the model is to attach a classifier to the *final* transformer layer, and then to jointly update both the backbone model and the classifier. The classifier is a one-layer fully-connected network. It takes as input the final layer hidden state $\mathbf{h}_n$ and outputs a prediction, which is a probability distribution over all classes for classification tasks. To enable early exiting, we instead attach a classifier $g_i(\cdot)$ to *every* transformer layer, i.e., there are n classifiers in total. Each classifier can make its own prediction, e.g., for the ith layer,

$$\hat{y}_i = g_i(\mathbf{h}_i; w_i), \tag{3.2}$$

where w_i is the parameters of the ith classifier. In this way, the model can accelerate inference by exiting earlier.

Fine-tuning For fine-tuning on a downstream task, the loss function of the ith classifier is

$$L_i(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} H(y, \hat{y}_i), \tag{3.3}$$

where $\mathcal{D}$ is the fine-tuning training set, (x, y) the feature–label pair of an example, $\mathbf{h}_i$ the ith layer's hidden state (generated from x), and H the cross-entropy loss function. We use a two-stage method to fine-tune the network: First, we update all transformer layers and the last classifier with the loss function L_n; this stage is identical to BERT fine-tuning in the original paper (Devlin et al., 2019). Second, we freeze all parameters fine-tuned in the first stage, and then update all but the

Algorithm 1 DeeBERT's entropy-based early exiting inference (input: x)

```
for i = 1 to n do
    ŷ_i = g_i(h_i; w_i)
    if entropy(ŷ_i) < S then
        return ŷ_i
    end if
end for
return ŷ_n
```

last classifier with the loss function $\sum_{i=1}^{n-1} L_i$. The reason for freezing parameters of transformer layers is to keep the optimal output quality for the last classifier; otherwise, transformer layers are no longer optimized solely for the last classifier, which generally worsens model quality. The objective functions can be written as

$$\text{Stage 1: } \min_{\substack{\theta_1,\cdots,\theta_n \\ w_n}} L_n; \tag{3.4}$$

$$\text{Stage 2: } \min_{w_1,\cdots,w_{n-1}} \sum_{i=1}^{n-1} L_i. \tag{3.5}$$

This two-stage fine-tuning can also be regarded as starting from a fine-tuned model, adding extra classifiers to it, and fine-tuning only these new classifiers. In the case where fine-tuned models are already available, this only requires minimal extra effort to enable early exiting.

3.2.2 DeeBERT at Inference

The way DeeBERT works at inference time is shown in Algorithm 1. We quantify an intermediate classifier's confidence in its prediction using the *entropy* of the output probability distribution. When an input example x arrives at an intermediate classifier, the classifier compares the entropy of its output distribution $\hat{y}_i$ with a preset threshold S to determine whether the example should be returned here or sent to the next transformer layer.

It is clear from both intuition and experimentation that a larger S leads to a faster but less accurate model, and a smaller S leads to a more accurate but slower one. In our experiments, we vary S based on this principle.

We also explore using ensembles of multiple layers instead of a single layer for the classifier, but this does not bring significant improvements. The reason is that predictions from different layers are usually highly correlated, and a wrong prediction from an intermediate layer is unlikely to be "fixed" by the layers before it. Therefore, we stick to the simple yet efficient single-output layer strategy.

3.3 Experimental Setup

We apply DeeBERT to both BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019), and conduct experiments on six sequence classification datasets from the GLUE benchmark: SST-2, MRPC, QNLI, RTE, QQP, and MNLI (Section 2.3). Inference runtime measurements are performed on a single NVIDIA Tesla P100 GPU. Hyperparameters such as hidden state sizes, learning rates, fine-tuning epochs, and batch sizes are kept unchanged from the Huggingface Transformer library (Wolf et al., 2020). There is no early stopping and the checkpoint after full fine-tuning is chosen.

3.4 Experimental Results

3.4.1 Main Results

We vary DeeBERT's accuracy–efficiency trade-off[17] by setting different entropy thresholds S, and compare the results with other baselines in Table 3.1. Model quality is measured on the test splits, and the results are provided by the GLUE evaluation server. Efficiency is quantified with the wall-clock inference runtime, including both CPU and GPU runtime, on the entire test set, where examples are fed into the model one by one. For each run of DeeBERT on a dataset, we choose three entropy thresholds S based on accuracy–efficiency trade-offs on the development set, aiming to demonstrate the following cases: (1) the maximum runtime savings with minimal

[17] In the term *accuracy*–efficiency trade-off, we use accuracy to broadly refer to model quality metrics, including accuracy itself, precision/recall, F1 score, etc.—depending on the task.

	SST-2		MRPC		QNLI		RTE		QQP		MNLI-(m/mm)	
	Acc	Time	F_1	Time	Acc	Time	Acc	Time	F_1	Time	Acc	Time
BERT$_{\text{BASE}}$												
Baseline	93.6	36.72s	88.2	34.77s	91.0	111.44s	69.9	61.26s	71.4	145min	83.9/83.0	202.84s
DistilBERT	−1.4	−40%	−1.1	−40%	−2.6	−40%	−9.4	−40%	−1.1	−40%	−4.5	−40%
	−0.2	−21%	−0.3	−14%	−0.1	−15%	−0.4	−9%	−0.0	−24%	−0.0/−0.1	−14%
DeeBERT	−0.6	−40%	−1.3	−31%	−0.7	−29%	−0.6	−11%	−0.1	−39%	−0.8/−0.7	−25%
	−2.1	−47%	−3.0	−44%	−3.1	−44%	−3.2	−33%	−2.0	−49%	−3.9/−3.8	−37%
RoBERTa$_{\text{BASE}}$												
Baseline	94.3	36.73s	90.4	35.24s	92.4	112.96s	67.5	60.14s	71.8	152min	87.0/86.3	198.52s
LayerDrop	−1.8	−50%	-	-	-	-	-	-	-	-	−4.1	−50%
	+0.1	−26%	+0.1	−25%	−0.1	−25%	−0.6	−32%	+0.1	−32%	−0.0/−0.0	−19%
DeeBERT	−0.0	−33%	+0.2	−28%	−0.5	−30%	−0.4	−33%	−0.0	−39%	−0.1/−0.3	−23%
	−1.8	−44%	−1.1	−38%	−2.5	−39%	−1.1	−35%	−0.6	−44%	−3.9/−4.1	−29%

Table 3.1: Comparison between baseline (original BERT/RoBERTa), DeeBERT, and other acceleration methods. LayerDrop only reports results on SST-2 and MNLI. Time savings of DistilBERT and LayerDrop are estimated by reported model size reduction.

performance drop ($< 0.5\%$), (2) the sweet point (elbow point) of trade-offs, and (3) the runtime savings with moderate performance drop ($2\% - 4\%$). Chosen S values differ for each dataset.

We also visualize the trade-off in Figure 3.2. Each curve is drawn by interpolating a number of points, each of which corresponds to a different threshold S. Since this only involves a comparison between different settings of DeeBERT, the runtime is measured on the development set.

From Table 3.1 and Figure 3.2, we observe the following patterns:

- Despite differences in baseline performance, both models show similar patterns on all datasets: the performance (accuracy/F_1 score) stays (mostly) the same until runtime saving reaches a certain turning point, and then starts to drop gradually. The turning point typically comes earlier for BERT than for RoBERTa, but after the turning point, the performance of

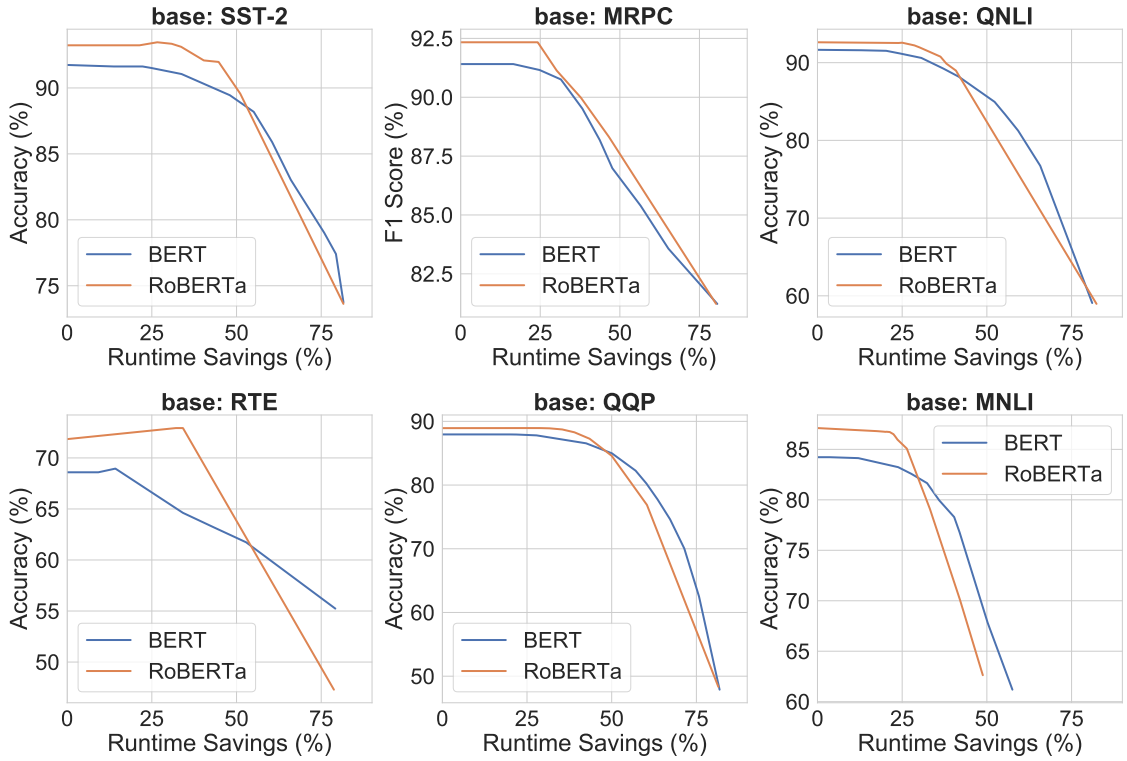


Figure 3.2: DeeBERT quality and efficiency trade-offs for $\text{BERT}_{\text{BASE}}$ and $\text{RoBERTa}_{\text{BASE}}$ models.

RoBERTa drops faster than for BERT. The reason for this will be discussed in Section 3.4.3.

- Occasionally, we observe spikes in the curves, e.g., RoBERTa in SST-2, and both BERT and RoBERTa in RTE. We attribute this to possible regularization brought by early exiting and thus smaller effective model sizes, i.e., in some cases, using all transformer layers may not be as good as using only some of them.

Compared with other BERT acceleration methods such as distillation and structured dropout, DeeBERT has the following two notable advantages:

- Instead of producing a fixed-size smaller model like DistilBERT (Sanh et al., 2019), DeeBERT produces a series of options for faster inference. This is equivalent to a sequence of models with increasing sizes where each model can reuse the computation of previous ones. Users have the flexibility to choose from these models according to their demands.

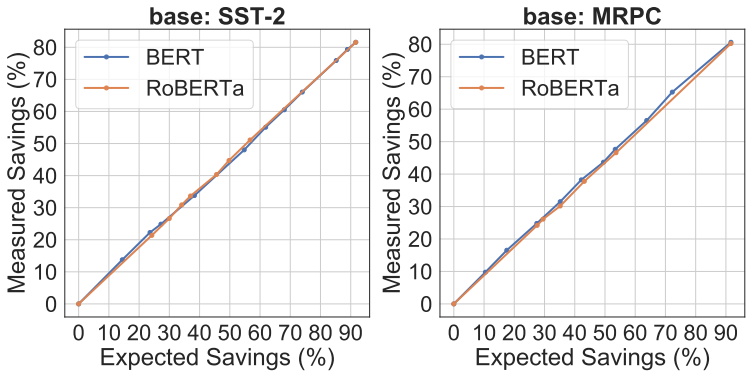


Figure 3.3: Comparison between expected saving (x-axis) and actual measured saving (y-axis), using $\text{BERT}_{\text{BASE}}$ and $\text{RoBERTa}_{\text{BASE}}$ models.

- Unlike DistilBERT and LayerDrop (Fan et al., 2020), DeeBERT does not require further pre-training of the transformer model, which is considerably more time-consuming than fine-tuning.

3.4.2 Expected Savings

As the measurement of runtime might not be stable, we propose another metric to capture efficiency, called expected saving, defined as

$$1 - \frac{\sum_{i=1}^{n} i \times N_i}{\sum_{i=1}^{n} n \times N_i}, \tag{3.6}$$

where n is the number of layers and N_i is the number of examples exiting at layer i. Intuitively, the expected saving is the fraction of transformer layer execution saved by using early exiting. The advantage of this metric is that it remains invariant between different runs and can be analytically computed. For validation, we compare this metric with measured saving in Figure 3.3. Overall, the curves show a linear relationship between expected savings and measured savings, indicating that our reported runtime is a stable measurement of DeeBERT's efficiency.

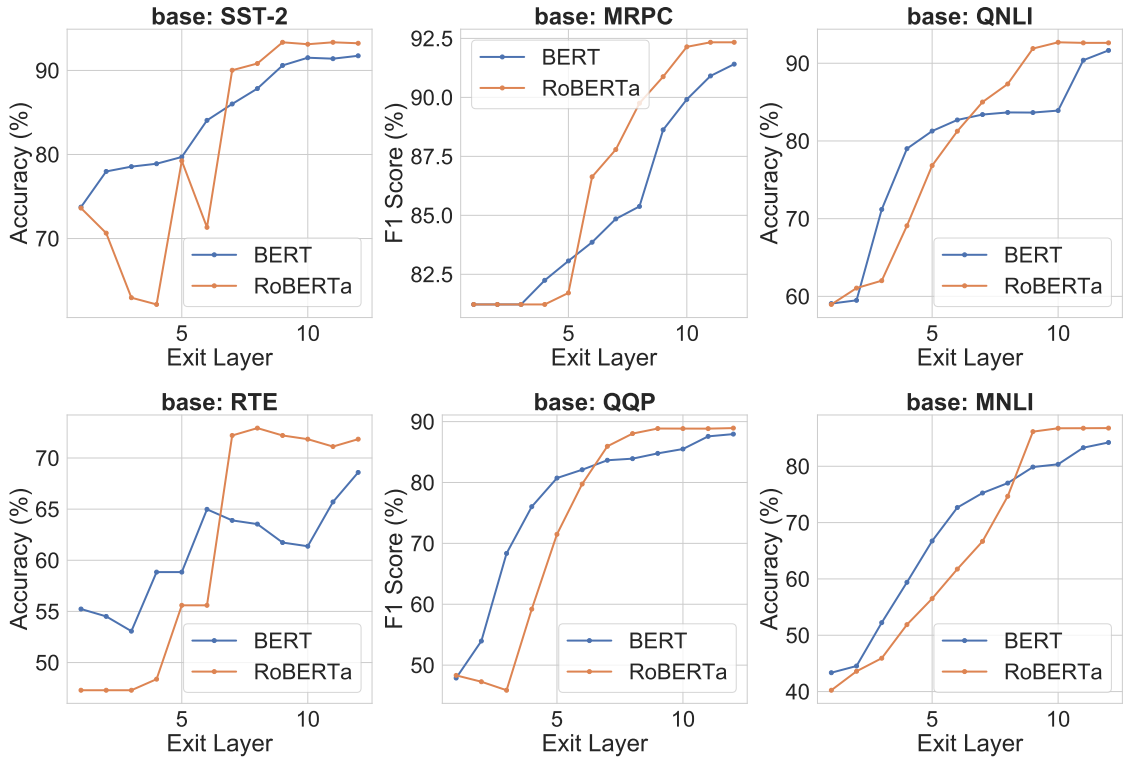


Figure 3.4: Accuracy of each classifier for $\text{BERT}_{\text{BASE}}$ and $\text{RoBERTa}_{\text{BASE}}$.

3.4.3 Layer-wise Analyses

In order to understand the effect of applying DeeBERT to both models, we conduct further analyses on each classifier layer. Experiments in this section are also performed on the development set.

Output Performance by Layer For each classifier, we force *all examples* in the development set to exit at the designated layer, measure the output quality, and visualize the results in Figure 3.4.

From the figure, we notice the difference between BERT and RoBERTa. The output quality of BERT improves at a relatively stable rate as the index of the exit classifier increases. The output quality of RoBERTa, on the other hand, stays almost unchanged (or even worsens) for a few layers, then rapidly improves, and reaches a saturation point before BERT does. This provides an explanation for the phenomenon mentioned in Section 3.4.1: on the same dataset, RoBERTa

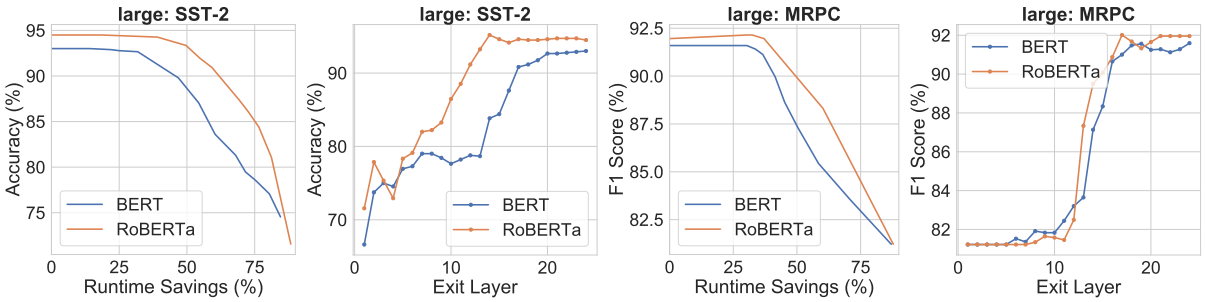


Figure 3.5: Results for $\text{BERT}_{\text{LARGE}}$ and $\text{RoBERTa}_{\text{LARGE}}$.

often achieves more runtime savings while maintaining roughly the same output quality, but then quality drops faster after reaching the turning point.

We also show the results for $\text{BERT}_{\text{LARGE}}$ and $\text{RoBERTa}_{\text{LARGE}}$ in Figure 3.5. From the two plots on the right, we observe signs of redundancy that both $\text{BERT}_{\text{LARGE}}$ and $\text{RoBERTa}_{\text{LARGE}}$ share: the last several layers do not show much improvement compared with the previous layers (performance even drops slightly in some cases). Such redundancy can also be seen in Figure 3.4.

Number of Exiting Examples by Layer We further show the fraction of examples exiting at each classifier for a given entropy threshold in Figure 3.6.

Entropy threshold $S = 0$ is the baseline, and all examples exit at the last layer; as S increases, gradually more examples exit earlier. Apart from the obvious, we observe additional, interesting patterns: if a layer does not provide better-quality output than previous layers, such as layer 11 in $\text{BERT}_{\text{BASE}}$ and layers 2–4 and 6 in $\text{RoBERTa}_{\text{BASE}}$ (which can be seen in Figure 3.4, top left), it is typically chosen by very few examples; popular layers are typically those that substantially improve over previous layers, such as layers 7 and 9 in $\text{RoBERTa}_{\text{BASE}}$. This shows that an entropy threshold is able to choose the fastest classifier among those with comparable quality, and achieves a good trade-off between quality and efficiency.

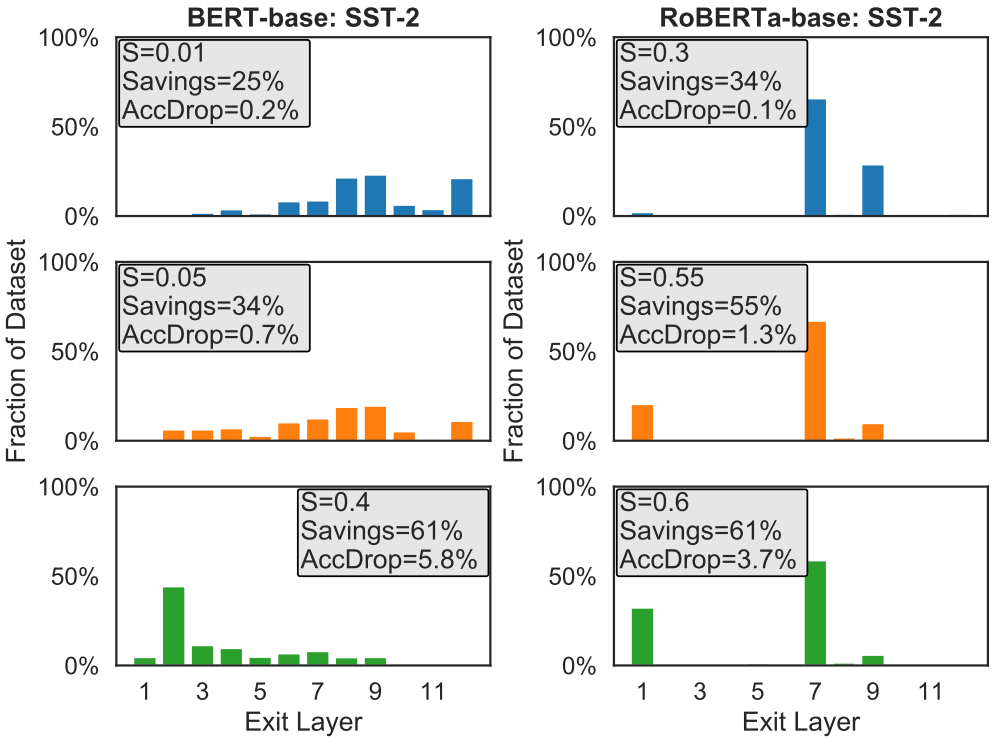


Figure 3.6: Number of output examples by layer for $\text{BERT}_{\text{BASE}}$ and $\text{RoBERTa}_{\text{BASE}}$. Each plot represents a separate entropy threshold S.

3.5 Summary

In this chapter, we discuss DeeBERT, our initial attempt to apply early exiting to transformer models for NLP. It is an effective method that exploits redundancy in BERT models to achieve better accuracy–efficiency trade-offs. Experiments demonstrate its ability to accelerate BERT's and RoBERTa's inference by up to $\sim$40%, and also reveal interesting patterns of different transformer layers in BERT models.

Chapter 4

Early Exiting for Sequence Regression and Low-Resource Datasets

In this chapter, we discuss two improvements over DeeBERT (Chapter 3). The results are published in the BERxiT (BERT + ExiT) paper (Xin et al., 2021a). The first improvement is to extend DeeBERT to regression tasks. DeeBERT and other similar work such as RTJ (Schwartz et al., 2020) design early exiting based on the assumption that the model's output is a probability distribution, which is not always the case. Inspired by explicit early exiting, we insert a learning-to-exit (LTE) module alongside the early exiting classifiers. The second improvement is a more sophisticated method for fine-tuning the early exiting model for low-resource datasets. DeeBERT's *two-stage* fine-tuning emphasizes the final layer's quality, while RTJ's *joint* fine-tuning emphasizes earlier layers' quality. We propose *alternating* fine-tuning which alternates the objective function between those of two-stage and joint fine-tuning, and thereby combining the advantages of both. More importantly, the improvement over joint fine-tuning is larger for low-resource datasets.

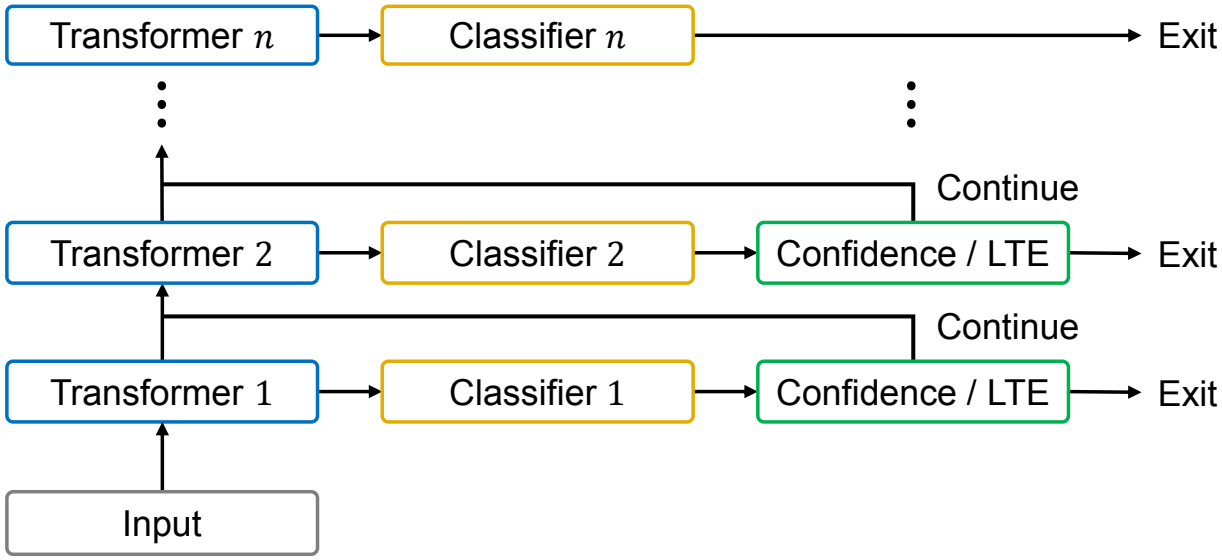


Figure 4.1: Multi-output structure of early exiting BERT. There are n layers in total; blue blocks are transformer layers, orange blocks are classifiers, and green blocks are exiting decision making modules. The green blocks are the difference between this diagram and Figure 3.1.

4.1 Overview

Early exiting (Schwartz et al., 2020; Xin et al., 2020a; Liu et al., 2020) has been proposed to accelerate the inference of BERT and models with similar architecture, i.e., those comprising multiple transformer layers (Vaswani et al., 2017) with a classifier at the top. Instead of using only one classifier, additional classifiers are attached to each transformer layer (see Figure 4.1), and the entire model is fine-tuned together. At inference time, the example can perform early exiting through one of the intermediate classifiers.

While previous early exiting papers provide promising accuracy–efficiency trade-offs, improvements are necessary for two important components: fine-tuning strategies and exiting decision making. In these papers, fine-tuning strategies are relatively simple and fail to take full advantage of the pre-trained model's effectiveness; we propose a novel fine-tuning strategy, *Alternating*, for this multi-output model. Moreover, previous work makes exiting decisions based on the confidence of output probability distributions, and is hence only applicable to classification

tasks; we extend it to other tasks by proposing the *learning-to-exit* idea. With carefully designed fine-tuning strategies and methods for making exiting decisions, the model can achieve better accuracy–efficiency trade-offs and can be extended to regression tasks.

We refer to these improvements collectively as BERxiT, and apply it to transformer-based models including BERT, RoBERTa (Liu et al., 2019), and ALBERT (Lan et al., 2020); we also apply it on top of another accelerated variant of BERT, DistilBERT (Sanh et al., 2019). We conduct experiments on datasets including classification and regression tasks, and show that our method can save up to 70% of inference time with minimal quality degradation. We also show that the early exiting idea is capable of providing insight into the inner mechanism of pre-trained models.

4.2 Model Structure and Fine-Tuning

The model structure, including the backbone model and classifiers, is the same as introduced in Section 3.2.1. The only difference is that for regression tasks, the output of classifiers is a scalar, but we still refer to these one-layer networks as *classifiers* for naming consistency.

We discuss how to fine-tune this multi-output network. To recap Equation (3.3), the loss function for the ith layer classifier is

$$L_i(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} H(y, \hat{y}_i), \tag{4.1}$$

where $\mathcal{D}$ is the fine-tuning training set, (x, y) the feature–label pair of an example, $\hat{y}_i$ layer's output (generated from x), and H the task-specific loss function, e.g., cross-entropy for classification tasks and mean squared error (MSE) for regression tasks.

The most straightforward fine-tuning strategy is perhaps minimizing the sum of all classifiers' loss functions and jointly updating all parameters in the process. We refer to this strategy as *Joint*, which is used in RTJ (Schwartz et al., 2020):

$$\min_{\substack{\theta_1,\cdots,\theta_n \\ w_1,\cdots,w_n}} \sum_{i=1}^{n} L_i. \tag{4.2}$$

If we hope to preserve the best model quality for the final layer, the desired fine-tuning strategy is *Two-stage*, which is used in DeeBERT (Chapter 3). The first stage is identical to vanilla BERT fine-tuning: updating the backbone model and only the final classifier. In the second stage, we freeze all parameters updated in the first stage, and fine-tune the remaining classifiers. Objectives in the two stages are as follows.

$$\text{Stage 1: } \min_{\substack{\theta_1,\cdots,\theta_n \\ w_n}} L_n \tag{4.3}$$

$$\text{Stage 2: } \min_{w_1,\cdots,w_{n-1}} \sum_{i=1}^{n-1} L_i \tag{4.4}$$

These two fine-tuning strategies are not ideal. Intuitively, in this multi-output network, loss functions of different classifiers interfere with each other in a *negative* way. Transformer layers have to provide hidden states for two competing purposes: immediate inference at the adjacent classifier and gradual feature extraction for future classifiers. Therefore, achieving a balance between the classifiers is critical. *Two-stage* produces final classifiers with optimal quality at the price of earlier layers, since most parameters are solely optimized for the final classifier. *Joint* treats all classifiers equally, and therefore its final classifier is less effective than that of *Two-stage*. To combine the advantages, we propose a novel fine-tuning strategy, *Alternating*. It alternates between two objectives (taken from Equation (4.2) and Equation (4.3)) for odd-numbered and even-numbered iterations.

$$\text{Odd: } \min_{\substack{\theta_1,\cdots,\theta_n \\ w_n}} L_n \tag{4.5}$$

$$\text{Even: } \min_{\substack{\theta_1,\cdots,\theta_n \\ w_1,\cdots,w_n}} \sum_{i=1}^{n} L_i \tag{4.6}$$

Combining objectives from *Joint* and *Two-stage*, *Alternating* has the potential to find the most preferable region in the parameter space, i.e., the intersection between optimal regions for different layers.

4.3 Exiting Decision Making

After the entire model (including the backbone and all classifiers) is fine-tuned, it can perform early exiting for an inference example. In this section, we discuss two methods to make exiting decisions.

4.3.1 Confidence Thresholding

When the model is "certain" enough of its prediction at an intermediate layer, the forward inference can be terminated, saving computation of the remaining layers.

For classification tasks, a straightforward measurement of the prediction certainty is the entropy of the output prediction, which is used in DeeBERT (Chapter 3). Similarly, we can also use the maximum entry of the output distribution, which we refer to as *confidence* and is used in previous work (Schwartz et al., 2020; Liu et al., 2020). Confidence yields similar results to entropy, and is also simpler and faster to calculate. Before inference starts, a confidence threshold is chosen. In forward propagation, the confidence of the output at each layer is compared with the threshold; if it is larger than the threshold at a certain layer, the example exits and future layers are skipped.

4.3.2 Learning to Exit

While using a confidence or entropy threshold is straightforward and effective, it is exploiting the fact that the classifier's output is a probability distribution in classification tasks. This is generally not the case for other tasks such as regression. To address the gap, we propose *learning-to-exit* (LTE) as a substitute when the distribution is unavailable.

The ith layer hidden state $\mathbf{h}_i$ is a vector in the embedding space. Intuitively, different regions of the embedding space have different certainty levels. For instance, in binary classification tasks, regions closer to the decision boundary have a lower certainty level, and this is explicitly expressed as a lower confidence of the output probability distribution. But even when certainty cannot be explicitly measured, we can still train an auxiliary LTE module to estimate such a metric by learning to distinguish different certainty levels in the embedding space.

Concretely, the LTE module is a simple one-layer fully-connected network. It takes as input the hidden state $\mathbf{h}_i$ and outputs the certainty level u_i of the example at the ith layer:

$$u_i = \sigma(\mathbf{c}^\top \mathbf{h}_i + b), \tag{4.7}$$

where σ is the sigmoid function, $\mathbf{c}$ is the weight vector, and b is the bias term.

The loss function for the LTE module is a simple MSE between u_i and the "ground truth" certainty level at the ith layer $\tilde{u}_i$:

$$J_i = ||u_i - \tilde{u}_i||_2^2. \tag{4.8}$$

For classification, the ground truth certainty level is whether the classifier makes the correct prediction:

$$\tilde{u}_i = \mathbb{1}[\arg\max_j \hat{y}_i^{(j)} = y], \tag{4.9}$$

where $\hat{y}_i$ is the output probability distribution at the ith layer and $\hat{y}_i^{(j)}$ is its jth entry. For regression, the ground truth certainty level is negatively related to the prediction's absolute error:

$$\tilde{u}_i = 1 - \tanh(|\hat{y}_i - y|), \tag{4.10}$$

where the $\tanh$ functions ensures that $\tilde{u}_i$ is in the range $[0, 1]$. In both cases, $\tilde{u}_i$ is larger when the predicted value of the model is closer to the ground truth label.

To apply LTE, we initialize the LTE module together with classifiers and it is shared among all layers. We train the LTE module jointly with the rest of the model by *substituting* L_i in Equations (4.2) to (4.6) with $L_i + J_i$. At inference time, we use the certainty level predicted by the LTE module as a proxy for confidence: if the predicted certainty level is higher than the chosen threshold, the inference example performs early exiting.

4.4 Experimental Setup

We evaluate different fine-tuning strategies and early exiting methods on BERT models. We conduct experiments on the following dataset: RTE, MRPC, SST-2, QNLI, QQP, MNLI, STS-B, and SICK (Section 2.3).

	RTE	MRPC	SST-2	QNLI	QQP	MNLI	STS-B	SICK
$\text{BERT}_{\text{BASE}}$	5.8	8.4	18.0	110.4	856.8	209.4	33.2	107.8
$\text{BERT}_{\text{LARGE}}$	11.6	17.4	35.9	223.2	1952.8	400.3	61.3	209.8
$\text{ALBERT}_{\text{BASE}}$	6.5	9.4	18.5	114.6	864.8	204.8	31.6	104.2
DistilBERT	3.0	4.3	9.1	55.4	407.1	106.2	15.9	50.9

Table 4.1: Inference runtime in seconds for each model and dataset.

We conduct grid search on experimental settings such as the optimizer, learning rates, hidden state sizes, and dropout probabilities, and discover that it is best to keep original settings from the Huggingface Transformers library (Wolf et al., 2020). Random seeds are also unchanged for fair comparisons. For BERT, ALBERT, and DistilBERT, we fine-tune for 3 epochs; for RoBERTa, we fine-tune for 10 epochs; no early-stopping or checkpoint selection is performed.

Experiments are done on a single NVIDIA P100 GPU. For inference, we use a batch size of 1 (since we need to perform early exiting based on each individual example's difficulty). Inference runtime for the entire dev set for all models and datasets is shown in Table 4.1. RoBERTa has the same model structure as BERT, and therefore its runtime is also very close to that of BERT. Note that this is affected by the computing environment and may vary between different runs. Numbers of parameters for BERT and ALBERT backbone models can be found in the paper by Lan et al. (2020). RoBERTa shares the same model structure as BERT and has the same number of parameters. Numbers of parameters for early-exiting-specific modules, such as additional classifiers and the LTE module, are on the order of thousands, and are therefore negligible compared with those of backbone models (millions).

Most results in this chapter use the dev split, since the large number of evaluations we need are forbidden by the GLUE evaluation server. The only exception is Table 4.2, where we report model accuracy–efficiency trade-offs on the test split.

4.4.1 Efficiency Metrics

While DeeBERT mainly uses measured wall-clock runtime as the efficiency metrics (Chapter 3), in this chapter we instead use *average exit layer* of all inference examples. This is equivalent to the *expected savings* mentioned in Section 3.4.2. We choose this new metric for the following three reasons.

It is linear w.r.t. the amount of computation. Inference time computation in our model occurs in the following parts: the embedding layer, transformer layers, classifiers, and the LTE module (if used). If a layer is chosen, i.e., the exit layer is after it, all components of the layer (transformer, classifier, LTE module) are used and incur computation costs. Additionally, embedding look-up (selecting a column in the matrix) is much faster than the above components (involving matrix-vector multiplication), and can therefore be neglected.

It is stable across different runs. With a fine-tuned model, an inference example's exit layer only depends on the confidence (or LTE-predicted certainty) at each layer and the threshold. On the other hand, direct measurement of wall-clock runtime is frequently affected by competing processes and fluctuates between different runs.

The computation overhead of early exiting is negligible. With the above reasons, there is only one concern left for using the average exit layer as the efficiency metric: how do additional layers in our model (including the additional classifier and possibly the LTE module) compare with transformer layers in the original BERT paper? We estimate FLOPS used in one example's inference as follows. Since we will eventually end up with orders of magnitude differences, we use the *big-theta* asymptotic notation for estimation.

Most computation is incurred for matrix and vector multiplication. Using the naïve implementation, the cost for multiplying two vectors in $\mathbb{R}^d$ is $\Theta(d)$, and the cost for multiplying a matrix in $\mathbb{R}^{d_1 \times d_2}$ and a vector in $\mathbb{R}^{d_2}$ is $\Theta(d_1 d_2)$.

We denote d as the hidden state size of our model (768 for base models and 1024 for large models), c as the number of classes (less than 4 in our experiments), n as the sequence length (typically in the hundreds), and h as the number of heads in multi-head attention (12 for base and 16 for large).

The classifier is a one-layer fully-connected layer, mapping a vector in $\mathbb{R}^d$ to an output in $\mathbb{R}^c$, therefore the cost is $\Theta(cd)$. Similarly, the cost of the LTE module is $\Theta(2d)$, since its output is always a vector in $\mathbb{R}^2$.

The transformer layer mainly consists of multi-head self-attention, an FFN layer, and two layer normalization modules. Layer normalization is much faster than the other two and we therefore neglect it. The FFN layer maps n vectors from $\mathbb{R}^d$ to $\mathbb{R}^d$, therefore the cost is $\Theta(nd^2)$. The multi-head attention[18] computes h individual uni-head attention. For each uni-head attention, the mapping from original query, key, and value vectors ($\mathbb{R}^d$) to head-specific ones ($\mathbb{R}^{d/h}$) incurs a cost of $\Theta(nd^2/h)$; calculating the attention results incurs a cost of $\Theta(nd/h)$. Therefore the total cost here is $\Theta(nd^2 + nd) = \Theta(nd^2)$. Finally, the results of each head are combined and one more matrix–vector multiplication is needed, incurring a cost of $\Theta(nd)$. The total cost of one transformer layer is therefore $\Theta(nd^2)$.

Comparing the above, the ratio of a transformer layer to a classifier/LTE module is

$$\frac{\Theta(nd^2)}{\Theta(cd + 2d)} = \Theta(nd). \tag{4.11}$$

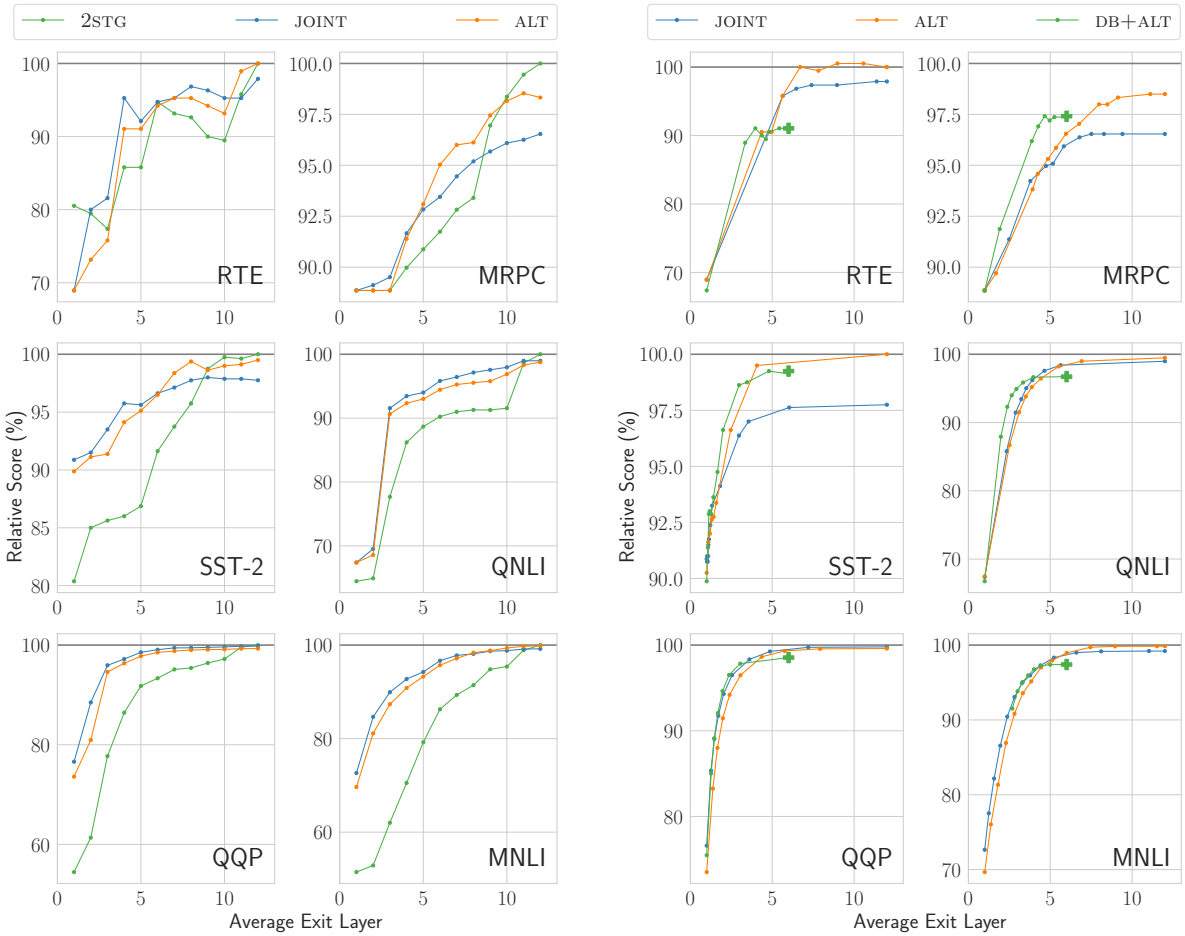
Considering the typical value of n and d (both are in the order of hundreds), the classifier and the LTE module are several orders of magnitude lighter than the transformer layer. Even with advanced algorithms and parallel hardware that may accelerate transformer layers, we can still safely come to the conclusion that the computation overhead of early exit is negligible.

4.5 Experimental Results

4.5.1 Layer-wise Scores Comparison

We discuss three fine-tuning strategies in Section 4.2: *Joint* (also used in RTJ), *Two-stage* (also used in DeeBERT), and *Alternating* (proposed in this chapter). In tables and figures, they are labeled respectively as JOINT, 2STG, and ALT. We compare these three fine-tuning strategies

[18]Details can be found in the paper by Vaswani et al. (2017).



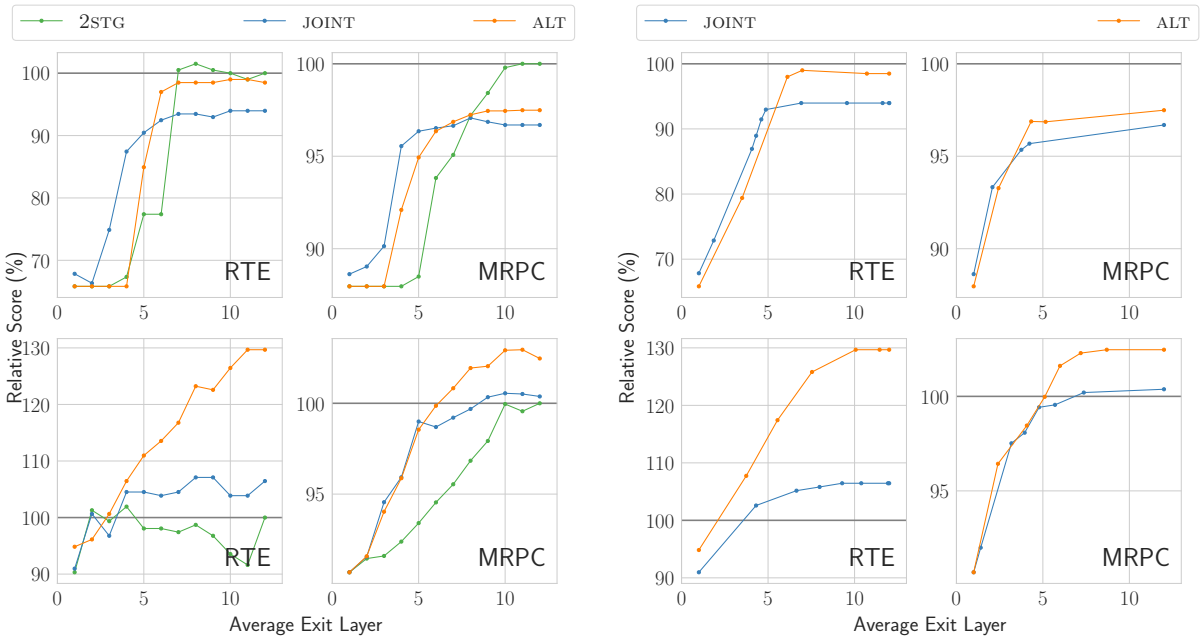
(a) Layer-wise relative scores of the three fine-tuning strategies on classification datasets.

(b) Accuracy–efficiency trade-offs using confidence for exiting decisions.

Figure 4.2: Results for $\text{BERT}_{\text{BASE}}$.

in Figure 4.2a and Figure 4.3a, respectively for $\text{BERT}_{\text{BASE}}$and $\text{RoBERTa}_{\text{BASE}}$/$\text{ALBERT}_{\text{BASE}}$.[19] We

[19]While we present the results on all datasets for $\text{BERT}_{\text{BASE}}$, we only present those for low-resource datasets for $\text{RoBERTa}_{\text{BASE}}$/$\text{ALBERT}_{\text{BASE}}$, since the differences are the most notable for low-resource ones (as we will show later).



(a) Layer-wise relative scores of the three fine-tuning strategies on classification datasets.

(b) Accuracy–efficiency trade-offs using confidence for exiting decisions.

Figure 4.3: Results for $\text{RoBERTa}_{\text{BASE}}$ (top row) and $\text{ALBERT}_{\text{BASE}}$ (bottom row).

show each fine-tuning strategy's *layer-wise score* curves: each point in the curve shows the output score at a certain exit layer, i.e., all examples are required to exit at this layer for evaluation. More specifically, we report *relative* scores, and the 100% baseline is the original score of the vanilla models without early exiting, which is also the score of the final layer of *Two-stage* because of parameter freezing in its second stage.

We observe the following from the figures:

- *Two-stage* is unsatisfying. While it achieves the best score at the final layer, it comes at a large cost of other layers, especially for non-low-resource datasets.
- *Alternating* is better than *Joint* in later layers, and weaker in earlier layers. However, as we will see in the next section, when we evaluate accuracy–efficiency trade-offs of confidence-

based early exiting, the weakness of *Alternating* in earlier layers is no longer substantial, while its advantage is preserved.

- The difference between *Joint* and *Alternating* is larger for low-resource datasets, where the training set is insufficient to fine-tune all layers well simultaneously.
- Interestingly, for ALBERT$_{\text{BASE}}$, *Alternating*'s relative scores are higher than 100% in the final layers. We speculate that this is because of the parameter-sharing nature of ALBERT and the small sizes of the datasets: better supervision for intermediate layers also helps the final layer.

4.5.2 Early Exiting Trade-offs Comparison

From the previous section, we see that *Two-stage* is visibly less preferable than the other two. Therefore in this section, we focus on the comparison between *Joint* and *Alternating* by showing their accuracy–efficiency trade-offs when confidence thresholding is used for making exiting decisions.

We visualize the trade-offs between *Joint* and *Alternating* in Figure 4.2b and Figure 4.3b. We also compare *Alternating* and other baselines by showing detailed numbers in Table 4.2 using results from the test set. Dots in the figures and ALT rows in the table are generated by varying the confidence threshold, and the thresholds are chosen to show trade-offs at different average exit layers. In addition to the comparison between *Joint* and *Alternating*, we add another strong baseline, DistilBERT (Sanh et al., 2019), to BERT$_{\text{BASE}}$'s results in Figure 4.2b. We apply *Alternating* fine-tuning and early exiting on top of DistilBERT (labeled as DB+ALT), and the rightmost point of the curve is DistilBERT itself without early exiting (the green ✚). Observations from the table and figures are as follows:

- On the test set (Table 4.2), early exiting with *Alternating* fine-tuning saves a large amount of inference computation, with only minimal quality degradation, compared with vanilla inference.

	RTE		MRPC		SST-2		QNLI		QQP		MNLI-(m/mm)		STS-B	
	Score	Layer	Score	Layer	Score	Layer	Score	Layer	Score	Layer	Score	Layer	Score	Layer
							BERT$_{\text{BASE}}$							
RAW	66.4	12	88.9	12	93.5	12	90.5	12	71.2	12	84.6/83.4	12	85.8	12
ALT	101%	−44%	99%	−30%	98%	−65%	99%	−42%	99%	−56%	99%/99%	−37%	95%	−50%
	99%	−54%	97%	−56%	96%	−79%	98%	−63%	97%	−75%	97%/97%	−57%	91%	−67%
	96%	−64%	94%	−74%	94%	−87%	95%	−71%	93%	−84%	93%/92%	−72%	85%	−75%
DB	86%	−50%	97%	−50%	98%	−50%	97%	−50%	98%	−50%	97%/97%	−50%	94%	−50%
DB+ALT	86%	−55%	97%	−60%	98%	−75%	97%	−72%	98%	−76%	96%/97%	−66%	93%	−66%
							BERT$_{\text{LARGE}}$							
RAW	70.1	24	89.3	24	94.9	24	92.7	24	72.1	24	86.7/85.9	24	86.5	24
ALT	95%	−33%	99%	−32%	100%	−32%	97%	−62%	98%	−74%	99%/99%	−36%	97%	−39%
	94%	−46%	98%	−46%	99%	−61%	95%	−73%	96%	−82%	96%/97%	−57%	90%	−62%
	88%	−62%	94%	−71%	96%	−78%	91%	−83%	91%	−89%	90%/90%	−75%	76%	−80%

Table 4.2: Test set results comparing baselines (raw BERT$_{\text{BASE}}$/BERT$_{\text{LARGE}}$, from the original paper), DistilBERT, and early exiting with *Alternating* fine-tuning. Metric for model quality: score for RAW baselines and relative scores for others. Metric for model efficiency: total layers for RAW; relative saved layers for others (w.r.t. raw models).

- Compared with *Joint*, *Alternating* inherits its benefits discussed in the previous section: better trade-offs at higher scores (larger average exit layer). Additionally, its improvements are larger in smaller datasets.
- *Alternating*'s weakness at more aggressive exiting (smaller average exit layer) is minimized. Take Figure 4.2b as an example, we report *the area of one curve above the other* as a numerical metric: JOINT over ALT and ALT over JOINT is respectively (0.4, 13.5) for RTE, (0.9, 8.8) for MRPC, and (0.2, 18.2) for SST-2. The advantage of *Alternating* indicates that later layers intrinsically contribute more to early exiting performance, partly because the final layer's score is the upper bound for all previous layers (ignoring randomness in training). This shows that the *Joint* fine-tuning strategy, which treats all layers equally, is not ideal.

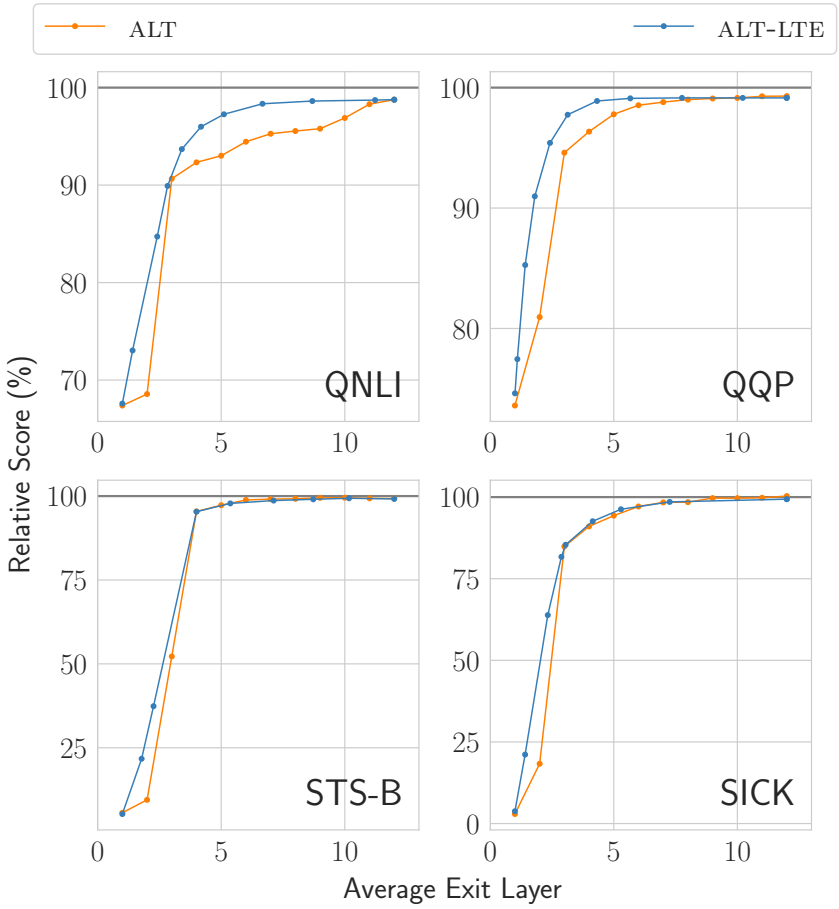


Figure 4.4: Comparing layer-wise score of *Alternating* with LTE-based early exiting on top of *Alternating*. $\text{BERT}_{\text{BASE}}$ is the backbone.

- In most cases, *Alternating* outperforms DistilBERT, which requires distillation in pre-training and is therefore much more resource-demanding. It also further improves model efficiency on top of DistilBERT, suggesting that early exiting can be further combined with other acceleration methods (see Chapter 7).

Method	Relative Score	Speedup	Avg. Exit Layer
PABEE	99%	2.1	5.7
	98%	2.4	5.0
ALT-LTE	99%	2.2	5.4
	98%	2.6	4.6

Table 4.3: Comparing LTE with PABEE on STS-B.

4.5.3 Learning to Exit Performance

To examine the effectiveness of LTE, we apply it on top of models fine-tuned with *Alternating*. We show the results in Figure 4.4 on four datasets, 2 classification and 2 regression.

We use the *layer-wise* score of *Alternating* as the baseline: if we want to save $x\%$ inference runtime, a straightforward way is to use the first $(100 - x)\%$ layers for every example, regardless of its difficulty. LTE is expected to dynamically allocate resources based on an example's difficulty and therefore outperform this baseline. From the figures for QNLI and QQP, we observe that the blue curves are substantially above the orange curves, i.e., LTE provides better accuracy–efficiency trade-offs than the layer-wise baseline, achieving the same model quality with less computation. For regression tasks STS-B and SICK, the layer-wise baseline reaches its maximum score at relatively early layers, leaving little room for LTE to perform. Nevertheless, LTE still outperforms the baseline, especially in earlier layers (note that the scale of y-axis is larger for regression tasks—from 0 to 100%).

In Table 4.3, we also compare LTE with the patience-based baseline PABEE (Zhou et al., 2020), which comes out at roughly the same time as BERxiT. We show their speedups and average exit layers at the same relative scores. PABEE does not provide exact speedup numbers; therefore we estimate the values from their figures. We can see that *Alternating* fine-tuning plus LTE is marginally better than PABEE on regression tasks.

We further compare LTE-predicted certainty for each layer with layer-wise scores in Figure 4.5. We observe large differences of predicted certainty both within and across layers, showing that

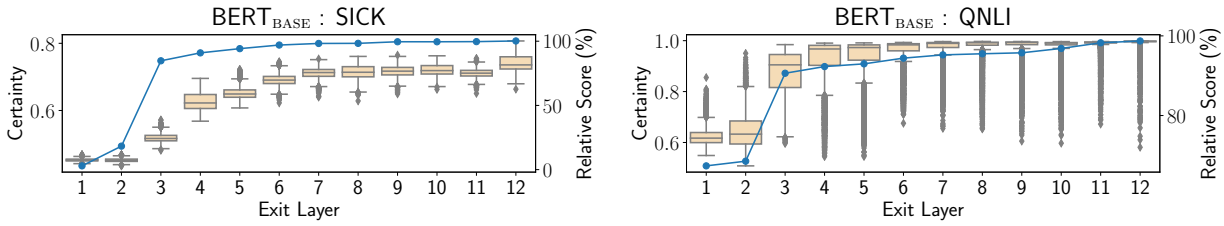


Figure 4.5: Comparison of each layer's score and learned certainty. Yellow box-plots: distribution of certainty at each layer; blue curve: relative score.

the LTE module is able to learn not only layer-specific information, but also example differences within the same layer. Also, predicted certainty is generally positively correlated with scores. This further demonstrates that the LTE module successfully captures certainty information based on the model's hidden state.

LTE extends confidence-based early exiting to tasks other than classification. Furthermore, our LTE module is more straightforward and intuitive than DAT (Elbayad et al., 2020), yet achieves comparable results with classification tasks.

4.5.4 Prediction Confidence as a Probe

So far, we have only regarded prediction confidence as something produced by the black-box model and used it for making early exiting decisions. In this section, we show an example of how confidence is related to a human-interpretable feature, demonstrating its potential to reveal the inner mechanisms of transformer models.

We choose two datasets, MRPC and QQP, where the task is to predict whether two input sequences are semantically equivalent. Intuitively, the BLEU score (Papineni et al., 2002) between the two sequences, which measures n-gram matching, may be related to the prediction. At each output layer, we first divide all dev set examples into two subsets by whether they are predicted as positive or negative; then, we calculate the BLEU-4 score for each example, and calculate the Pearson correlation between BLEU scores and confidence in each subset; finally, we compare the correlation for both subsets in each layer, along with the layer-wise relative scores,

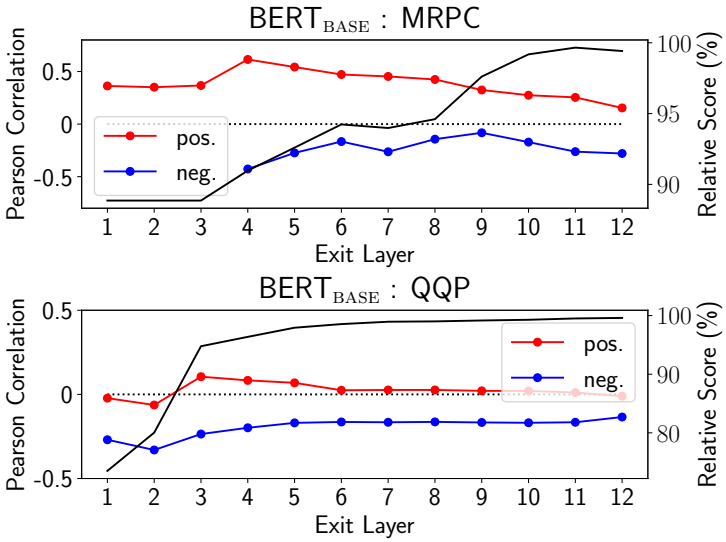


Figure 4.6: Comparison between BLEU–confidence correlation (red and blue) and layer-wise scores (black).

in Figure 4.6.

We notice that the BLEU scores and predicted confidence show the strongest correlation in layers where the model quality starts to improve (layer 4–5 in MRPC[20] and 2–3 in QQP). After these layers, the correlation gradually weakens. It suggests that in the early layers, the model relies more on simple features such as n-gram matching for making semantic judgments: the higher the BLEU score is, the more certain it is for making positive predictions and the less certain it is for negative ones; however, with more layers, the model acquires the ability to look beyond the BLEU score, reducing its reliance on n-gram matching and achieving better performance. Therefore, with MRPC as an example, analyzing differences between layers 3 and 4 may reveal how the model detects n-gram matching, and analyzing differences between layers 4 and 6 may reveal advanced semantic features learned by the model.

[20]In MRPC, the first three layers only make positive predictions due to the highly imbalanced training label distribution.

4.6 Summary

To improve early exiting for BERT, we present BERxiT, including the *Alternating* fine-tuning strategy which outperforms methods from previous papers and the LTE idea which extends early exiting to a broader range of tasks. Experiments show the effectiveness of *Alternating* in providing better accuracy–efficiency trade-offs and the successful application of LTE to regression tasks. They also show that early exiting is cumulative with other acceleration methods such as DistilBERT and has the potential for model interpretation.

Chapter 5

Early Exiting for Document Reranking

We apply early exiting to BERT for document reranking and publish the results in the Early Exiting MonoBERT paper (EEMB) (Xin et al., 2020b). This is of particular interest, not only because reranking is an important task of the second stage of the retrieve–process pipeline, but also because it is fundamentally different from sequence classification tasks in GLUE (Chapter 3) and requires special design in early exiting.

5.1 Overview

In this chapter, we study how to accelerate the inference of BERT-based document reranking models. We follow the framework of MonoBERT (Nogueira and Cho, 2019), which performs binary classification on query–document pairs into relevant/non-relevant. To accelerate inference for BERT, we employ the idea of *early exiting* as in DeeBERT (Xin et al., 2020a). Here, extra classification layers are attached to transformer layers of a pre-trained BERT model (Figure 5.1). The model is then fine-tuned on the downstream training dataset. At inference time, an example is sequentially processed by transformer layers and classifiers. If a classifier is confident of its prediction, it returns the result and inference ends early; otherwise, the next transformer layer proceeds with the computation. Different from DeeBERT, which treats all classes equally, we use *asymmetric* early exiting for document ranking: the exiting threshold for positive predictions is

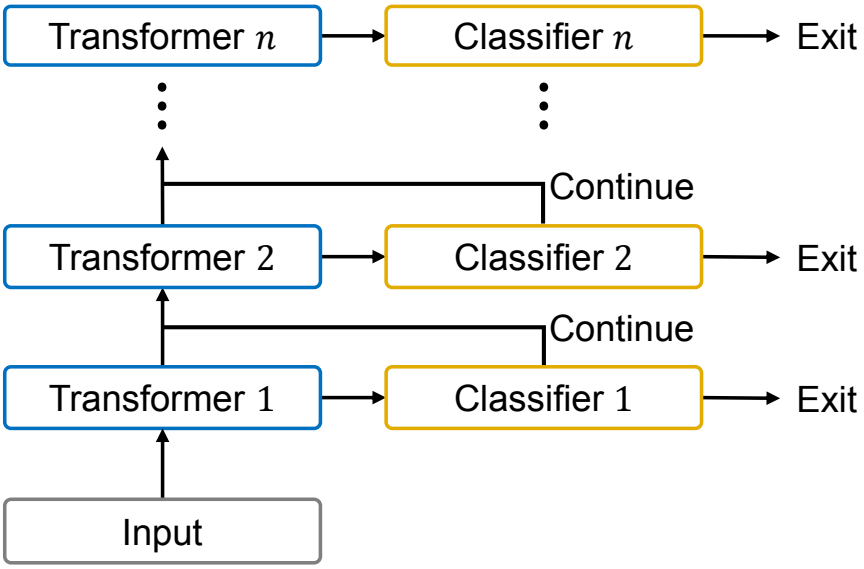


Figure 5.1: Overview of early exiting BERT for document ranking. Blue blocks are transformer layers and orange blocks are classifiers. The model architecture is the same as Figure 3.1.

higher than for negative ones, since the two classes in document ranking are intrinsically different, and it is natural to allocate more computational resources for positive examples.

We conduct experiments on $\text{BERT}_{\text{BASE}}$ with two document ranking datasets, MS MARCO passage (Bajaj et al., 2016) and ASNQ (Garg et al., 2019). We compare against Cascade Transformer (Soldaini and Moschitti, 2020), a recently proposed technique to accelerate inference in BERT-based document ranking. Results show that our method can reduce inference latency by up to 2.5× with minimal effectiveness degradation.

5.2 Early Exiting for Document Ranking

The task of concern is *document re-ranking*, i.e., to rank among a small candidate document set, which is generated by a faster but less accurate "bag of words" IR technique such as BM25. We assume in this chapter that the candidate set is provided as input.

5.2.1 MonoBERT

We first briefly describe MonoBERT (Nogueira and Cho, 2019), the neural reranking model on which our early exiting model is built.

The input to MonoBERT is a query–document pair (Q, D), which is organized as one input sequence in the following format, as described in Section 2.1.2:

$$\texttt{[CLS]}\ Q\ \texttt{[SEP]}\ D\ \texttt{[SEP]}$$

The task of MonoBERT is technically binary classification: it produces a probability distribution over two classes, relevant and non-relevant.

MonoBERT is initialized with a pre-trained BERT model (or other models with a similar architecture such as RoBERTa). A classifier, which is typically a single-layer fully-connected network, is attached to the last transformer layer of the BERT model; concretely, the classifier takes as input the last layer hidden state corresponding to the `[CLS]` token, and outputs the binary prediction ($\hat{y}_n$ in Equation (3.2)). For fine-tuning, the model is updated with binary label supervision. For inference, a query–document pair's relevance score is the predicted probability of the document being relevant, and this score is used for subsequent re-ranking of the candidates.

5.2.2 Fine-Tuning Early Exiting MonoBERT

Our model, early exiting MonoBERT, is a multi-output variant of BERT which enables early exiting. Similar to MonoBERT, we start with a pre-trained $\text{BERT}_{\text{BASE}}$ model with n transformer layers and attach n classifiers to it (Figure 5.1). A classifier is an FFN layer that outputs a probability distribution over two classes. To recap Section 3.2.1, at the ith layer, the hidden state and the output probability distribution are

$$\mathbf{h}_i = f_i(x; \theta_1, \cdots, \theta_i), \tag{5.1}$$

$$\hat{y}_i = g_i(\mathbf{h}_i; w_i) = (\hat{y}_i^{\text{pos}}, \hat{y}_i^{\text{neg}}), \tag{5.2}$$

and the loss function is

$$L_i(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} H(y, \hat{y}_i). \tag{5.3}$$

We simply use the *Joint* fine-tuning strategy described in Section 4.2, since with abundant training data that the document ranking task typically provides, this simple fine-tuning method yields good results and is also the fastest:

$$\min_{\substack{\theta_1,\cdots,\theta_n \\ w_1,\cdots,w_n}} \sum_{i=1}^{n} L_i. \tag{5.4}$$

5.2.3 Asymmetric Early Exiting

After the multi-output model is fine-tuned, it is used for inference with early exiting. When an inference example is fed into the model, it is processed sequentially by each transformer layer and classifier. If the ith layer classifier is confident of the prediction $\hat{y}_i$, early exiting is performed and subsequent transformer layers are skipped. We define the *confidence* of $\hat{y}_i$ as the higher probability of the two classes, $\hat{y}_i^{\text{pos}}$ and $\hat{y}_i^{\text{neg}}$. Finally, documents are ranked according to their predicted probability of the positive class $\hat{y}_i^{\text{pos}}$.

In previous early exiting BERT for NLP papers, *symmetric* early exiting is used, i.e., if the model's confidence exceeds a threshold, the example exits. The early exiting algorithm is therefore symmetric with respect to all classes. In the case of early exiting for document ranking, however, there are two fundamental differences from NLP applications. Firstly, the two classes (relevant and non-relevant) are clearly not symmetric: we only care about relevant documents, and the more relevant they are, the more computation resources should be allocated to them. Secondly, for positive examples, confidence is not only the criterion for early exiting, but also the score for subsequent re-ranking.

To bridge the differences mentioned above, we propose to use *asymmetric* early exiting for document ranking. Concretely, we define two thresholds for confidence, S_p and S_n, for positive (relevant) and negative (non-relevant) predictions, respectively. We will show by experiments that we should choose a higher positive confidence threshold than the negative one, i.e., if a document is likely to be non-relevant, then we can stop its inference earlier, but if it is expected to be relevant, we should be prudent and use more layers to obtain accurate scores. Details are shown in Algorithm 2.

Algorithm 2 Asymmetric Early Exiting

for $i = 1$ to n **do**
  $\hat{y}_i = g_i(\mathbf{h}_i; w_i)$
  if $\hat{y}_i^{\text{pos}} > S_p$ **or** $\hat{y}_i^{\text{neg}} > S_n$ **then**
    return $\hat{y}_i^{\text{pos}}$
  end if
end for
return $\hat{y}_n^{\text{pos}}$

5.3 Experimental Setup

We apply early exiting on $\text{BERT}_{\text{BASE}}$ and conduct experiments on two datasets for document ranking, MS MARCO passage (Bajaj et al., 2016) and ASNQ (Garg et al., 2019). We fine-tune the model on 4 NVIDIA Tesla V100 GPUs, with a batch size of 60. For other hyperparameters such as learning rate and maximum sequence length, we follow MonoBERT (Nogueira and Cho, 2019) for MS MARCO passage and Cascade Transformer (Soldaini and Moschitti, 2020) for ASNQ.

MS MARCO passage provides a small version of the training set,[21] from which we build our training set by selecting tuples with *unique* pairs of query and relevant document, yielding a dataset of 832k query–document pairs, half relevant and half non-relevant. We fine-tune the model for 4 epochs. Its development set has 6.9k queries, and for each query there are 1k candidate documents, among which there is approximately 1 relevant document. Its development set has 6.9k queries, and for each query there is approximately 1 relevant document. For re-ranking, we use the top 1k candidates generated by BM25 ranking from the Anserini toolkit (Yang et al., 2017).

ASNQ's training set[22] has 20M query–document pairs. Similar to MS MARCO passage, we select only unique pairs of query and relevant document, and then complement the dataset with the same amount of pairs of query and non-relevant document, yielding a training set of 114k pairs. We fine-tune for 2 epochs. Its development set has 1.3k queries, and each query has, on

[21] https://msmarco.blob.core.windows.net/msmarcoranking/triples.train.small.tar.gz
[22] https://github.com/alexa/wqa_tanda

Method	MRR @10	S_n	Speedup
MB	0.347		1.0×
eeMB	0.343	1.00	1.0×
	0.343 (−0%)	0.95	2.6×
	0.340 (−1%)	0.90	2.9×
	0.336 (−2%)	0.85	3.2×
	0.327 (−5%)	0.80	3.5×
	0.312 (−9%)	0.75	3.9×
	0.290 (−15%)	0.70	4.3×

Table 5.1: MS MARCO passage development set results. MB: MonoBERT; eeMB: early exiting MonoBERT.

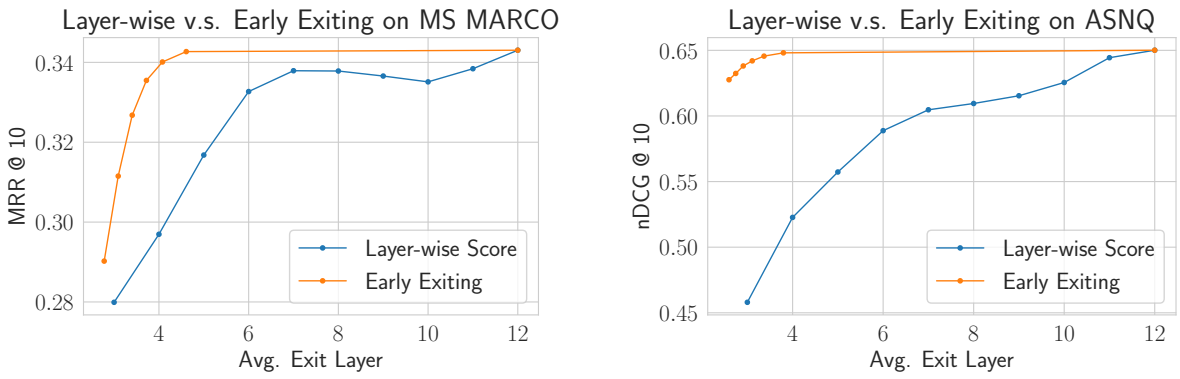
Method	nDCG @10	MRR	S_n	Speedup
CT	0.661	0.654		1.0×
	0.653	0.653		1.6×
	0.650	0.648		1.8×
	0.650	0.645		2.0×
eeMB	0.650	0.633	1.00	1.0×
	0.650	0.633	0.99	2.5×
	0.648	0.632	0.95	3.2×
	0.646	0.632	0.90	3.6×
	0.638	0.627	0.80	4.1×

Table 5.2: ASNQ development set results. CT: Cascade Transformer; eeMB: early exiting MonoBERT. Absolute values of nDCG and MRR scores are *not directly comparable* between CT and eeMB, due to differences in dataset preparation.

average, 400 candidate documents and 3 relevant ones.

5.4 Experimental Results

We show the trade-offs between model quality and computation of early exiting in Table 5.1 and Table 5.2 for MS MARCO passage and ANSQ, respectively. Different trade-offs are achieved by setting various negative confidence thresholds S_n, while S_p is always set to 1; we will provide detailed analyses of these thresholds later. Inference efficiency is quantified by the average exit layer of inference examples; this metric is, according to our experiments, proportional to actual wall-clock runtime, while being invariant across multiple runs (please refer to Section 3.4.2 and Section 4.4.1 for details).



(a) MS MARCO passage development set

(b) ASNQ development set.

Figure 5.2: Comparison between layer-wise scores and early exiting trade-offs.

We can see that in both datasets, early exiting is able to accelerate inference by $\sim$2.5$\times$ while maintaining the original model effectiveness. It is worth noting that in Cascade Transformer (CT) (Soldaini and Moschitti, 2020), only a part of the development set is used for evaluation, and therefore the scores are not directly comparable. However, in terms of *relative* performance, our model appears to achieve higher inference speedup with a comparable score degradation. On the other hand, CT supports inference with multiple examples per batch, while our implementation of eeMB supports single-example inference only. It is our future plan to enable batch inference for early exiting.

We also compare confidence-based early exiting trade-offs with *layer-wise scores* of the model in Figure 5.2a and Figure 5.2b. The layer-wise score of the ith layer is obtained by forcing *all* inference examples to exit through the classifier at the ith layer. It provides a series of baselines: if we want to save 50% inference computation for a 12-layer model, a straightforward way is to use the 6th layer's classifier for all examples. The first two layers are omitted from the layer-wise score curves since their scores are too low to be useful. The figures show that the early exiting idea significantly outperforms the naïve baselines. On the other hand, it also provides us with an estimate of how many layers are necessary to achieve comparable accuracy as the full model. In our experiments, it takes about 4 layers, which is 1/3 of the full model. This aligns well with results from Table 5.1 and Table 5.2.

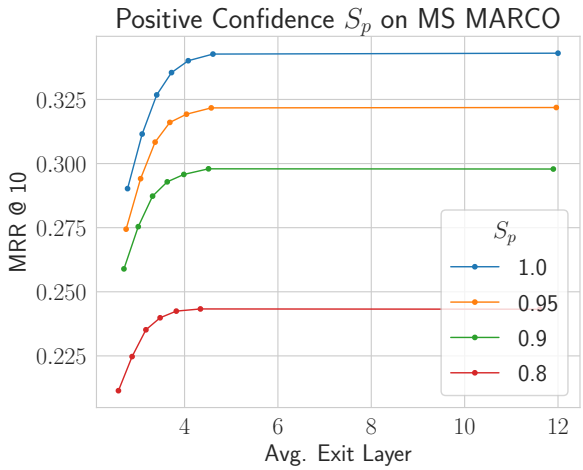


Figure 5.3: Comparison between different values of S_p (curves) and S_n (points on a curve).

To analyze the effect of different confidence thresholds, we plot in Figure 5.3 the comparison of confidence thresholds S_p and S_n using the MS MARCO passage dataset. Results from ASNQ are very similar and therefore omitted. Each curve corresponds to one S_p value, and points within a curve are plotted by choosing different S_n values.[23]

We notice that the trade-off performance is monotonic with respect to the positive confidence threshold: the higher S_p is, the better the trade-offs are. We speculate the reason is that while smaller S_p improves efficiency by allowing more examples to exit earlier, the quality of predictions from earlier layers of relevant examples degrades drastically. Considering the fact that relevant examples constitute only a tiny fraction of all candidates in both datasets, setting $S_p = 1$, i.e., using as many transformer layers as we can on positive examples, is the optimal choice. On the other hand, within one curve (a fixed S_p), S_n controls the trade-offs between efficiency and quality: lower S_n allows more negative examples to exit earlier, thus improving efficiency with relatively small quality degradation. Such asymmetry between S_p and S_n demonstrates the necessity of using two confidence thresholds instead of one: while we can safely perform early exiting on negative examples and save computation, we should allocate more resources to positive examples for more accurate predictions.

[23]Points on each curve, from left to right, correspond to S_n values in Table 5.1, from bottom to top.

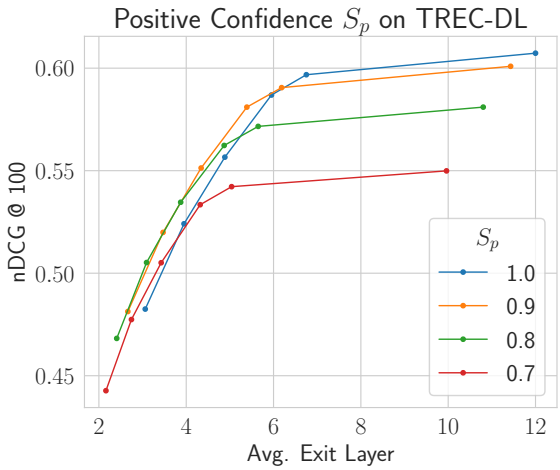


Figure 5.4: Comparison between different values of S_p (curves) and S_n (points on a curve). This is similar to Figure 5.3, but using a dataset with a much higher relevant document proportion.

It is worth noting that for datasets with extremely high relevant candidate proportions, 1.0 may no longer by the optimal choice for S_p. We demonstrate such a case by applying the MS MARCO passage checkpoint on the TREC 2019 Deep Learning Passage test set (TREC-DL) (Craswell et al., 2020).[24] On average, each query has about 100 relevant documents, and we consider the extreme case of reranking the top 200 documents.[25] We use nDCG@100 as the effectiveness metric, and the results are shown in Figure 5.4. Because relevant documents now constitute about a half of the candidates, we can see that setting $S_p = 1$ is no longer the absolute optimal choice; in fact, there is hardly a single optimal value for S_p. This example shows that while setting $S_p = 1$ is a good idea most of the time, we need to make proper adjustments for extreme cases.

[24] This dataset uses the same passage collection as MS MARCO passage, and the queries are also similar. Therefore it is reasonable to reuse checkpoints from MS MARCO.

[25] These top documents are retrieved with BM25, using Pyserini (Lin et al., 2021a).

5.5 Summary

We propose asymmetric early exiting BERT for document ranking, an effective method to improve model efficiency in IR tasks. Computation resources are allocated to examples according to their needs. Experiments show that our method is able to achieve different quality–latency trade-offs by setting different thresholds, and also improves model efficiency over baselines.

Chapter 6

Selective Prediction for NLP Tasks

The idea of early exiting can be concisely summarized as—we make the prediction as soon as we are confident. On the other side of the spectrum, another natural idea is—we abstain from making any prediction at all, if we are not confident. This is the setting of selective prediction (El-Yaniv and Wiener, 2010; Geifman and El-Yaniv, 2017; Moon et al., 2020). We study this problem in the context of NLP models and tasks, and publish the results in the Art of Abstention paper (Xin et al., 2021b).

6.1 Overview

Pre-trained language models based on the transformer architecture (Vaswani et al., 2017) have improved the state-of-the-art results in many NLP applications. Naturally, these models are deployed in various real-world applications. However, one may wonder whether they are always reliable, as pointed out by Guo et al. (2017) that modern neural networks, while having better accuracy, tend to be overconfident compared to simple networks from 20 years ago.

In this chapter, we study the problem of selective prediction (Geifman and El-Yaniv, 2017) in NLP. Under the setting of selective prediction, a model is allowed to *abstain* from making predictions on uncertain examples (Figure 6.1) and thereby reduce the error rate. This is a practical setting in a lot of realistic scenarios, such as making entailment judgments for breaking

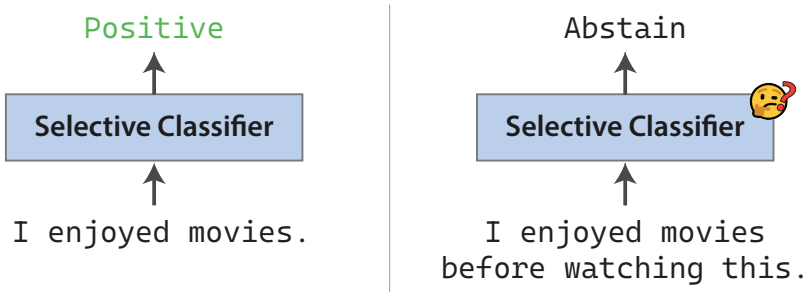


Figure 6.1: Example of a selective classifier that makes a prediction for a confident example (left) and abstains for an uncertain one (right).

news articles in search engines (Carlebach et al., 2020) and making critical predictions in medical and legal documents (Zhang et al., 2019b). In these cases, it is totally acceptable, if not desirable, for the models to admit their uncertainty and call for help from humans or better (but more costly) models.

Under the selective prediction setting, we construct a selective classifier by pairing a standard classifier with a confidence estimator. The confidence estimator measures how confident the model is for a certain example, and instructs the classifier to abstain on uncertain ones. Naturally, a good confidence estimator should have higher confidence for correctly classified examples than incorrect ones. We consider two choices of confidence estimators, softmax response (SR) (Hendrycks and Gimpel, 2017), and Monte-Carlo dropout (MC-dropout) (Gal and Ghahramani, 2016). SR directly interprets the output of the final softmax layer as a probability distribution and the highest probability as confidence. MC-dropout repeats the inference process multiple times, each time with a different dropout mask, and treats the negative variance of maximum probability as confidence. Confidence estimation is critical to selective prediction, and therefore studying this problem also helps relevant tasks such as active learning (Cohn et al., 1995; Shen et al., 2018) and early exiting (Schwartz et al., 2020; Xin et al., 2020a; Zhou et al., 2020; Xin et al., 2021a).

In this chapter, we compare selective prediction performance of different NLP models and confidence estimators. We also propose a simple trick, error regularization, which can be applied to any of these models and confidence estimators, and improve their selective prediction

performance. We further study the application of selective prediction on a variety of interesting applications, such as classification with no valid labels (no-answer problem) and using classifier cascades for accuracy–efficiency trade-offs. Experiments show that recent powerful NLP models such as BERT (Devlin et al., 2019) and ALBERT (Lan et al., 2020) improve not only accuracy but also selective prediction performance; they also demonstrate the effectiveness of the proposed error regularization by producing better confidence estimators which reduce the area under the *risk–coverage curve* by 10%.

6.2 Background

We introduce relevant concepts about selective prediction and confidence estimators, using multi-class classification as an example.

6.2.1 Selective Prediction

Given a feature space $\mathcal{X}$ and a set of labels $\mathcal{Y}$, a standard classifier f is a function $f : \mathcal{X} \to \mathcal{Y}$. A *selective classifier* is another function $h : \mathcal{X} \to \mathcal{Y} \cup \{\perp\}$, where $\perp$ is a special label indicating the abstention of prediction. Normally, the selective classifier is composed of a pair of functions $h = (f, g)$, where f is a standard classifier and g is the *selective function* $g : \mathcal{X} \to \{0, 1\}$. Given an input $x \in \mathcal{X}$, the output of the selective classifier is as follows:

$$h(x) = \begin{cases} f(x), & \text{if } g(x) = 1, \\ \perp, & \text{if } g(x) = 0, \end{cases} \tag{6.1}$$

and we can see that the output of g controls prediction or abstention. In most cases, g consists of a *confidence estimator* $\tilde{g} : \mathcal{X} \to \mathbb{R}$, and a *confidence threshold* θ:

$$g(x) = \mathbb{1}[\tilde{g}(x) > \theta]. \tag{6.2}$$

$\tilde{g}(x)$ indicates how confident the classifier f is on the example x, and θ controls the overall prediction versus abstention level.

A selective classifier makes trade-offs between *coverage* and *risk*. Given a labeled dataset $S = \{(x_i, y_i)\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}$ and an error function $\mathcal{L}$ to calculate each example's error $l_i = \mathcal{L}(f(x_i), y_i)$, the coverage and the selective risk of a classifier $h = (f, g)$ on S are, respectively,

$$\gamma(h) = \frac{1}{|S|} \sum_{(x_i, y_i) \in S} g(x_i), \tag{6.3}$$

$$r(h) = \frac{\sum_{(x_i, y_i) \in S} g(x_i) l_i}{\sum_{(x_i, y_i) \in S} g(x_i)}. \tag{6.4}$$

The selective classifier aims to minimize the selective risk at a given coverage.

The performance of a selective classifier $h = (f, g)$ can be evaluated by the risk–coverage curve (RCC) (El-Yaniv and Wiener, 2010), which is drawn by varying the confidence threshold θ (see Figure 6.2 for an example). Quantitatively, the *area under curve* (AUC) of RCC measures the effectiveness of a selective classifier.[26]

In order to minimize the AUC of RCC, the selective classifier should, intuitively, output $g(x) = 1$ for correctly classified examples and $g(x) = 0$ for incorrect ones. Therefore, an ideal $\tilde{g}$ has the following property: $\forall (x_i, y_i), (x_j, y_j) \in S$, $\tilde{g}(x_i) \leq \tilde{g}(x_j)$ iff $l_i \geq l_j$. We propose the following metric, *reversed pair proportion* (RPP), to evaluate how far the confidence estimator $\tilde{g}$ is to ideal, given the labeled dataset S of size n:

$$\text{RPP} = \frac{\sum_{1 \leq i,j \leq n}^{n} \mathbb{1}[\tilde{g}(x_i) < \tilde{g}(x_j), l_i < l_j]}{n^2}. \tag{6.5}$$

RPP measures the proportion of example pairs with a reversed confidence–error relationship, and the n^2 in the denominator is used to normalize the value. An ideal confidence estimator has an RPP value of 0.

6.2.2 Confidence Estimators

In most cases of deep learning for multi-class classification, the last layer of the classifier is a softmax activation, which outputs a probability distribution $P(y)$ over the set of labels $\mathcal{Y}$, where

[26] AUC in this chapter always corresponds to RCCs.

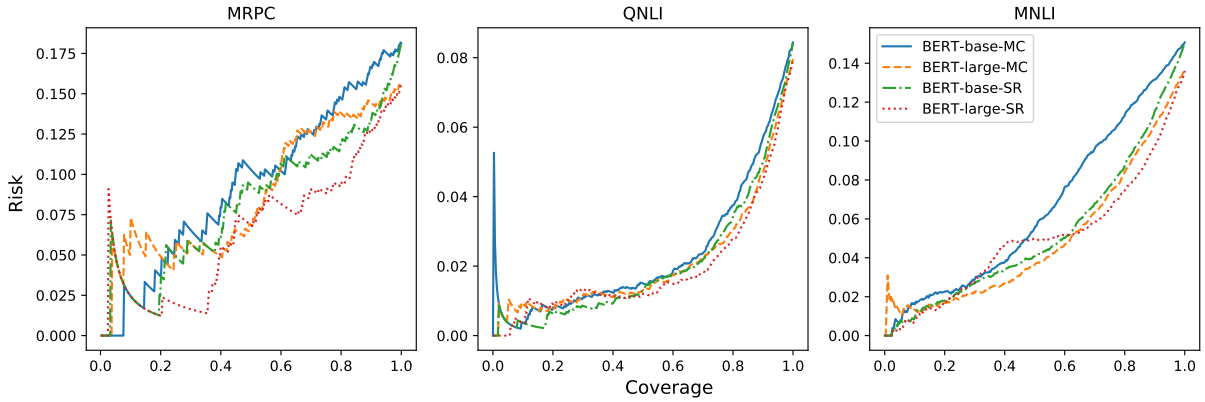


Figure 6.2: Risk–coverage curves of $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$ models with SR and MC confidence estimators. The legend applies to all sub-plots.

$y \in \mathcal{Y}$ is a label. In this case, the classifier can be written as

$$f(x) = \hat{y} = \arg\max_{y \in \mathcal{Y}} P(y), \tag{6.6}$$

where $\hat{y}$ is the label with the highest probability.

Perhaps the most straightforward and popular choice for the confidence estimator is softmax response (Hendrycks and Gimpel, 2017):

$$\tilde{g}_{\text{SR}}(x) = P(\hat{y}) = \max_{y \in \mathcal{Y}} P(y). \tag{6.7}$$

Alternatively, we can use the difference between probabilities of the top two classes for confidence estimation. We refer to this method as PD (probability difference).

Gal and Ghahramani (2016) argue that "softmax outputs are often erroneously interpreted as model confidence", and propose to use MC-dropout as the confidence estimator. In MC-dropout, $P(\hat{y})$ is computed for a total of R times, using a different dropout mask at each time, producing $P_1(\hat{y}), P_2(\hat{y}), \cdots, P_R(\hat{y})$. The variance of them is used to estimate the confidence:

$$\tilde{g}_{\text{MC}}(x) = -\text{Var}[P_1(\hat{y}), \cdots, P_R(\hat{y})]. \tag{6.8}$$

The negative sign is used here because a larger variance indicates a greater uncertainty, i.e., a lower confidence (Geifman and El-Yaniv, 2017; Kamath et al., 2020). By using different

dropout masks, MC-dropout is equivalent to using an ensemble for confidence estimation, but does not require actually training and storing multiple models. Nevertheless, compared to SR, the inference cost of MC-dropout is multiplied by R, which can be a problem when model inference is expensive.

6.3 Error Regularization

6.3.1 Regularizers

SR and MC-dropout are often used directly out of the box as the confidence estimator. Instead, we propose a simple regularization trick that can be easily applied at training (or fine-tuning for pre-trained models) time and can improve the effectiveness of the induced confidence estimators.

Considering that a good confidence estimator should minimize RPP defined in Equation (6.5), we add the following regularizer to the original training loss function:

$$L_{\text{total}} = \sum_{i=1}^{n} H(f(x_i), y_i) + \lambda L_{\text{reg}}, \tag{6.9}$$

$$L_{\text{reg}} = \sum_{1 \le i,j \le n} \Delta_{i,j} \ \mathbb{1}[e_i > e_j], \tag{6.10}$$

$$\Delta_{i,j} = \max\{0, \tilde{g}_{\text{SR}}(x_i) - \tilde{g}_{\text{SR}}(x_j)\}^2. \tag{6.11}$$

Here, $H(\cdot, \cdot)$ is the task-specific loss function such as cross entropy (H is not the same with the error function $\mathcal{L}$), λ is the hyperparameter for regularization, $\tilde{g}_{\text{SR}}$ is the maximum softmax probability defined in Equation (6.7), and e_i is the error of example i at the current iteration—details to calculate it will be explained in the next paragraph. We use SR confidence here because it is easily accessible at training time, while MC-dropout confidence is not. The intuition of this regularizer is as follows: if the model's error on example i is larger than its error on example j (i.e., example i is considered more "difficult" for the model), then the confidence on example i should *not* be greater than the confidence on example j.

In practice, at each iteration of training (fine-tuning), we can obtain the error e_i in one of the two following ways.

- **Current iteration error** We simply use the error function $\mathcal{L}$ to calculate the error of the example at the current iteration, and use it as e_i. In the case of multi-class classification, $\mathcal{L}$ is often chosen as the 0–1 error.

- **History record error** Since we intend to use e_i to quantify how difficult an example is, we draw inspiration from *forgettable examples* (Toneva et al., 2019). We calculate example error with $\mathcal{L}$ throughout the training process, and use the error averaged from the beginning to the current iteration as e_i. In this case, e_i takes value from the range $[0, 1]$.

6.3.2 Practical Approximations

In practice, it is computationally prohibitive to either strictly compute L_{reg} from Equation (6.10) for all example pairs, or to calculate history record error after every iteration. We therefore make the following two approximations.

For L_{reg} from Equation (6.10), we only consider examples from the mini-batch of the current iteration. For current iteration error, where e_i takes value from $\{0, 1\}$, we consider all pairs (i, j) where $e_i = 1$ and $e_j = 0$. For history record error, where e_i takes value from $[0, 1]$, we sort all examples in the mini-batch by their errors, and divide the mini-batch into 20% of examples with high error values and 80% of examples with low error values;[27] then we consider all pairs (i, j) where example i is from the former 20% and j from the latter 80%.

For calculating history record error, we compute and record the error values for the entire training set 10 times per epoch (once after each 10% iterations). At each training iteration, we use the average of error values recorded so far as e_i.

6.4 Experimental Setup

We conduct experiments of selective prediction on NLP tasks. Since the formulation of selective prediction is model agnostic, we choose the following representative models: (1) $\text{BERT}_{\text{BASE}}$ and

[27] We choose this 20–80 division to mimic the current iteration error case, where roughly 20% of training examples have an error of 1 and 80% have an error of 0.

$\text{BERT}_{\text{LARGE}}$ (Devlin et al., 2019), the dominant transformer-based models of recent years; (2) $\text{ALBERT}_{\text{BASE}}$ (Lan et al., 2020), a variant of BERT featuring parameter sharing and memory efficiency; (3) Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), the popular pre-transformer model that is lightweight and fast.

In this section, we compare the performance of selective prediction of these models, demonstrate the effectiveness of the proposed error regularization, and show the application of selective prediction in two interesting scenarios—the no-answer problem and the classifier cascades.

We conduct experiments mainly on three datasets: MRPC, QNLI, and MNLI (Section 2.3). Following the setting of the GLUE benchmark (Wang et al., 2018a), we use the training set for training/fine-tuning and the development set for evaluation (the test set's labels are not publicly available); MNLI's development set has two parts, *matched* and *mismatched* (m/mm).

The transformers models are taken from pre-trained checkpoints by Huggingface Transformers (Wolf et al., 2020). The LSTM is randomly initialized without pre-training. It is a two-layer bi-directional LSTM, with a hidden size of 200. On top of it there is a max-pooling layer and a fully-connected layer.

Training/fine-tuning and inference are done on a single NVIDIA Tesla V100 GPU. Since we are evaluating the selective prediction performance of different models instead of pursuing state-of-the-art results, we do not extensively tune hyperparameters; instead, most experiment settings such as hidden sizes, learning rates, and batch sizes are kept unchanged from the Huggingface Library. More specifically, all these models are trained/fine-tuned for 3 epochs without early-stopping or checkpoint selection; the learning rate is 2×10^{-5}; a batch size of 32 is used for training/fine-tuning; the maximum input sequence length is 128; choices for the regularization hyperparameter λ from Equation (6.9) are shown in Table 6.1.

6.5 Experimental Results

6.5.1 Comparing Different Models

We compare selective prediction performance of different models in Table 6.2. For each model, we report the performance given by the two confidence estimators, softmax response (SR) and

Model	curr.	hist.
LSTM	0.5	0.5
$\text{BERT}_{\text{BASE}}$	0.05	0.05
$\text{BERT}_{\text{LARGE}}$	0.1	0.1
$\text{ALBERT}_{\text{BASE}}$	0.01	0.05

Table 6.1: Choices for λ for different models and regularization methods.

Model	Confidence Estimator	MRPC			QNLI			MNLI-(m/mm)		
		F1(↑)	AUC(↓)	RPP(↓)	Acc(↑)	AUC(↓)	RPP(↓)	Acc(↑)	AUC(↓)	RPP(↓)
LSTM	SR	81.8	101.5	9.0	62.7	1539.1	9.0	65.4/64.3	1984.0/1990.0	6.8/6.6
	MC		137.0	11.3		2039.8	11.6		3554.1/3548.1	11.7/11.7
BERT base	SR	87.7	33.8	3.7	91.6	111.9	1.2	84.9/84.6	514.8/491.7	2.6/2.4
	MC		38.3	4.5		130.1	1.3		639.0/677.5	3.4/3.5
BERT large	SR	89.0	27.0	3.2	92.0	105.6	1.1	86.4/86.0	486.1/470.0	2.5/2.4
	MC		35.9	4.3		114.6	1.2		482.0/510.2	2.5/2.6
ALBERT base	SR	90.9	16.0	2.1	90.9	122.8	1.2	84.7/85.4	469.3/453.5	2.3/2.3
	MC		43.9	5.6		160.0	1.6		921.1/878.3	5.0/4.7

Table 6.2: Comparing selective prediction performance of different models and confidence estimators. All metrics except AUC are in percentages. ↑: higher is better; ↓: lower is better.

MC-dropout (MC). Our preliminary experiments show that the results of using PD for confidence estimation are very similar to those of SR, and therefore we omit the results of PD. The accuracy and the F1 score[28] measure the effectiveness of the classifier f, RPP measures the reliability of the confidence estimator $\tilde{g}$, and AUC is a comprehensive metric for both the classifier and the confidence estimator. The choice of confidence estimator does not affect the model's accuracy. We also provide risk–coverage curves (RCCs) of different models and confidence estimators in Figure 6.2. MC in the table and the figure uses a dropout rate of 0.01 and repetitive runs $R = 10$.

We first notice that models with overall higher accuracy also have better selective prediction performance (lower AUC and RPP). For example, compared with LSTM, $\text{BERT}_{\text{BASE}}$ has higher

[28]We henceforth refer to both accuracy and F1 scores simply as *accuracy* for the sake of conciseness.

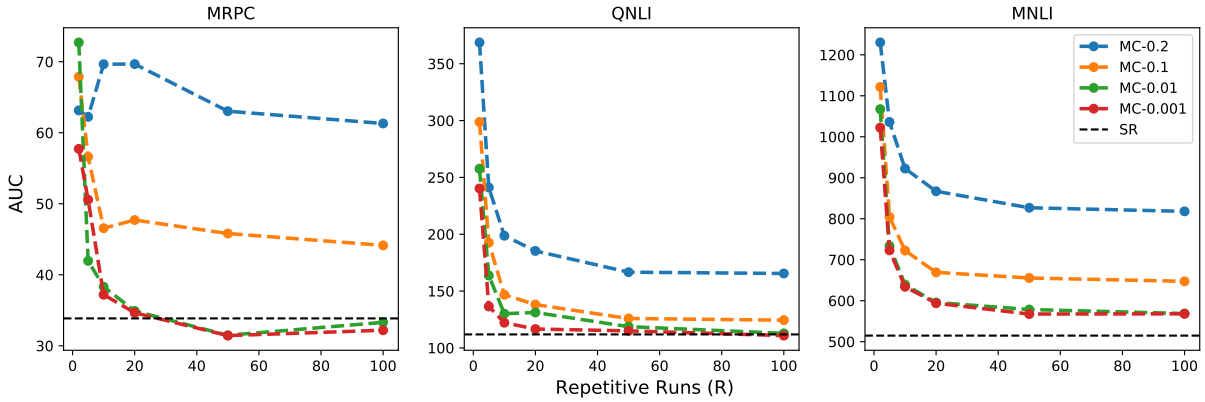


Figure 6.3: Selective prediction performance of MC-dropout with different numbers of repetitive runs (x-axis) and dropout rates (marked in the legend). $\text{BERT}_{\text{BASE}}$ is used here. The legend applies to all sub-plots.

accuracy and lower AUC/RPP on all datasets, and the same applies to the comparison between $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$. While AUC comprehensively reflects the effectiveness of both the classifier f and the confidence estimator $\tilde{g}$, RPP measures only the performance of $\tilde{g}$. Since the classifier's effectiveness does not directly affect RPP, the consistency of RPP's and accuracy's improvement indicates that sophisticated models simultaneously improve *both* model accuracy and confidence estimation. This is in contrast to the discovery by Guo et al. (2017) that sophisticated neural networks, despite having better accuracy, are more easily overconfident and worse calibrated than simple ones.

We also notice that MC-dropout performs consistently worse than softmax response, shown by both AUC and RPP. This shows that for NLP tasks and models, model confidence estimated by MC-dropout fails to align well with real example difficulty. We further study and visualize in Figure 6.3 the effect of different dropout rates and different numbers of repetitive runs R on MC-dropout's selective prediction performance. From the figure we can see that:

- For good MC-dropout performance, the dropout rate should not be too large. If it gets larger than 0.01, the performance worsens. On the other hand, further reducing the dropout rate does not lead to improvements.
- MC-dropout needs at least 20 repetitions to obtain results comparable to SR, which is

Model	Reg.	MRPC			QNLI			MNLI-(m/mm)		
		F1(↑)	AUC(↓)	RPP(↓)	Acc(↑)	AUC(↓)	RPP(↓)	Acc(↑)	AUC(↓)	RPP(↓)
LSTM	none	81.8	101.5	9.04	62.7	1539.1	8.99	65.4/64.3	1984.0/1990.0	6.81/6.60
	curr.	81.2	94.0	7.70	64.7	1376.2	8.51	65.5/64.4	1976.0/1990.5	6.80/6.64
	hist.	81.2	**92.3**	7.58	64.7	**1368.0**	8.42	65.3/64.6	**1974.4/1987.4**	6.74/6.71
BERT BASE	none	87.7	33.8	3.70	91.6	111.9	1.16	84.9/84.6	514.8/491.7	2.55/2.41
	curr.	88.1	31.2	3.49	91.9	**100.1**	1.08	84.6/84.6	**479.1/461.8**	2.39/2.27
	hist.	87.9	**30.3**	3.51	91.4	113.9	1.20	84.4/84.5	490.7/472.5	2.42/2.32
BERT LARGE	none	89.0	27.0	3.17	92.0	105.6	1.06	86.4/86.0	486.1/470.0	2.45/2.39
	curr.	89.7	**20.6**	3.05	91.2	**98.2**	1.04	86.5/85.5	417.7/434.4	2.17/2.17
	hist.	89.0	24.4	3.30	92.1	99.4	0.94	85.5/85.9	**404.9/400.6**	2.25/2.25
ALBERT BASE	none	90.9	16.0	2.13	90.9	122.8	1.21	84.7/85.4	469.3/453.5	2.32/2.30
	curr.	91.4	**13.2**	1.82	90.9	**104.3**	1.23	84.7/85.2	**451.2**/463.9	2.25/2.23
	hist.	91.0	16.2	2.18	91.2	117.5	1.12	84.6/85.2	461.1/**429.8**	2.26/2.30

Table 6.3: Comparing different regularizers (Reg.) for different models and datasets. Selective prediction performance is measured by AUC and RPP. All metrics except AUC are in percentages.

extremely expensive, especially for transformers.

Although MC-dropout has a sound theoretical foundation, its practical application to NLP tasks needs further improvements.

6.5.2 Effect of Error Regularization

In this part, we show that our simple regularization trick improves selective prediction performance. In Table 6.3, we report the accuracy, AUC, and RPP for each model, paired with three different regularizers: no regularization (none), current error regularizer (curr.), and history error regularizer (hist.), as described in Section 6.3.

We first see that applying error regularization (either current or history) does not harm model accuracy. There are minor fluctuations, but generally speaking, error regularization has no negative effect on the models' effectiveness.

We can also see that error regularization improves models' selective prediction performance, reducing AUC and RPP. As we mention in the previous section, AUC is a comprehensive metric for both the classifier f and the confidence estimator $\tilde{g}$. We therefore focus on this metric in this section, and we bold the lowest AUC in Table 6.3. We see that error regularization consistently achieve the lowest AUC values, and on average, the best scores are approximately 10% lower than the scores without regularization. This shows that error regularization produces confidence estimators that give better confidence rankings.

The two regularization methods, current error and history error, are similar in quality, with neither outperforming the other across all models and datasets. Therefore, we can conclude only that the error regularization trick improves selective prediction, but the best specific method varies. We leave this exploration for future work.

6.5.3 The No-Answer Problem

In this section, we conduct experiments to see how selective classifiers perform on datasets that either allow abstention or, equivalently, provide the *no-answer* label. This no-answer problem occurs whenever a trained classifier encounters an example whose label is unseen in training, which is common in practice. For example, in the setting of ultrafine entity typing with more than 10,000 labels (Choi et al., 2018), it is unsurprising to encounter examples with unseen types. Ideally, in this case, the classifier should choose the no-answer label. This setting is important yet often neglected, and there exist few classification datasets with the no-answer label. We therefore build our own datasets, binarized MNLI and SST-5 (bMNLI and bSST-5), to evaluate different models in this setting (Table 6.4).

The MNLI dataset is for sentence entailment classification. Given a pair of sentences, the goal is to predict the relationship between them, among three labels: entailment, contradiction, and neutral. The SST-5 dataset is for fine-grained sentence sentiment classification. Given a sentence, the goal is to predict the sentiment of it, among five labels: strongly positive, mildly positive, strongly negative, mildly negative, and neutral. To convert the original MNLI and SST-5 datasets into our binarized versions bMNLI and bSST-5, we modify the following: for SST-5, we merge strongly and mildly positive/negative into one positive/negative class; for MNLI, we simply regard entailment as positive and contradictory as negative. We then remove all neutral

Dataset	#Train	#Dev (m/mm)	#Labels
MRPC	3.7k	0.4k	2
QNLI	104.7k	5.5k	2
MNLI	392.7k	9.8k/9.8k	3
SST-5	8.5k	1.1k	5
bMNLI	261.8k	9.8k/9.8k	2+1
bSST-5	6.9k	1.1k	2+1

Table 6.4: Dataset statistics. bMNLI/bSST-5 are binarized version of MNLI/SST-5, with two normal labels and a special *no-answer* label.

instances from the training set but keep those in the development and test sets. Comparison between datasets used in the experiments is shown in Table 6.4. This way, in bMNLI and bSST-5, neutral instances in the development and test sets should be classified as no-answer by the model. A good model is expected to assign neutral examples in the development and test sets with *low confidence scores*, thereby predicting the *no-answer* label for them.

We report results for these two datasets with the no-answer label in Table 6.5. Accuracy (Acc), AUC, and RPP have the same meaning from the previous sections. We also consider a new metric specifically for the no-answer setting, *augmented accuracy* (Acc*), which is calculated as follows: (1) we make a number of attempts by searching a threshold α from 0.7 to 1.0 in increments of 0.01; (2) for each attempt, we regard all examples with predicted confidence lower than α as neutral, and then calculate the accuracy; (3) among all attempts, we take the highest accuracy as Acc*. Choosing the optimal α requires knowing the ground-truth answers in advance and is not practical in reality.[29] Instead, Acc* indicates how well a model recognizes examples whose label is likely unseen in the training set.

We first see that Acc* is consistently higher than Acc in all cases. This is unsurprising, but it demonstrates that unseen samples indeed have lower confidence and shows that introducing

[29]Alternatively, one may use a validation set to choose the optimal α. In our experiments, however, we use the development set for evaluation, since the labels of the test set itself are not publicly available. Holding out a part of the training set for validation is left for future exploration.

Model	Reg.	bSST5				bMNLI-(m/mm)			
		Acc(↑)	Acc*(↑)	AUC(↓)	RPP(↓)	Acc(↑)	Acc*(↑)	AUC(↓)	RPP(↓)
BERT BASE	none	71.7	74.0	174.7	5.34	63.9/64.2	70.6/70.9	1645.4/1630.7	4.74/4.72
	curr.	72.0	73.8	173.5	5.35	63.8/64.2	**71.1**/71.4	**1562.2/1562.1**	4.41/4.45
	hist.	72.6	**74.7**	**157.4**	5.17	63.8/64.1	70.7/**71.6**	1630.5/1583.2	4.62/4.51
BERT LARGE	none	73.2	73.7	158.4	5.58	64.8/64.8	72.9/72.5	1861.0/1852.3	5.18/5.16
	curr.	73.3	**74.2**	**137.1**	4.82	64.5/65.1	72.7/73.2	**1476.8**/1629.7	4.91/4.68
	hist.	73.5	73.7	148.8	4.53	65.0/64.8	**73.1/73.2**	1695.9/**1460.6**	4.14/4.11
ALBERT BASE	none	72.3	**73.5**	172.4	5.61	64.0/64.3	71.6/72.4	1579.1/1534.4	4.44/4.34
	curr.	72.4	73.2	168.0	5.32	63.8/64.2	**72.9**/72.3	**1563.8**/1550.9	4.45/4.32
	hist.	72.5	73.2	**161.0**	5.63	63.9/64.4	71.6/**73.5**	1601.8/**1496.3**	4.20/4.10

Table 6.5: Selective prediction performance of different models and regularization methods (Reg.) on two datasets with the *no-answer* label. All metrics except AUC are in percentages.

the abstention option is beneficial in the no-answer scenario. Also, we observe that error regularization improves the models' selective prediction performance, producing lower AUC/RPP and higher Acc* in most cases. This further demonstrates the effectiveness of the simple error regularization trick.

Secondly, we can see that the improvement of Acc* over Acc is larger in bMNLI than in bSST-5. The reason is that in bMNLI, neutral examples constitute about a third of the entire development set, while in bSST-5 they constitute only a fifth. The improvement is positively correlated with the proportion of neutral examples, since they are assigned lower confidence scores and provide the potential for abstention-based improvements.

6.5.4 Classifier Cascades

In this section, we show how confidence estimation and abstention can be used for accuracy–efficiency trade-offs. We use classifier cascades: we first use a less accurate classifier for prediction, abstain on examples with low confidence, then send them to more accurate but more costly classifiers. Here we choose LSTM and $\text{BERT}_{\text{BASE}}$ to constitute the cascade, but one can also choose other models and more levels of classifiers.

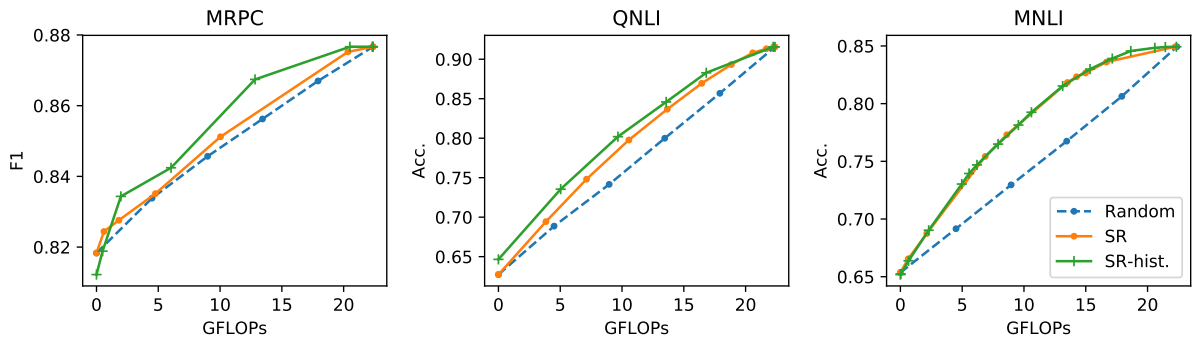


Figure 6.4: Accuracy–efficiency trade-offs by using classifier cascades. All examples are first evaluated by LSTM, and then we compare three ways of choosing examples to send to the more sophisticated model ($\text{BERT}_{\text{BASE}}$): random selection (Random), SR without regularization (SR), and SR with history error regularization (SR-hist.). The legend applies to all sub-plots.

We first use an LSTM for all examples' inference, and then send "difficult" ones to $\text{BERT}_{\text{BASE}}$. Since the computational cost of LSTM is negligible[30] compared to $\text{BERT}_{\text{BASE}}$, the key to efficiency here is correctly picking the "difficult" examples.

In Figure 6.4, we show the results of accuracy/F1 score versus average FLOPs[31] per inference example. Each curve represents a method to choose difficult examples: The blue curves are obtained by randomly selecting examples, as a simple baseline. The orange and green curves are obtained by using SR of LSTM as the indicator of example difficulty; the orange curves represent the LSTM trained with no regularization while the green curves are with history error regularization. Different points on the curves are chosen by varying the proportion of examples sent to the more accurate model, $\text{BERT}_{\text{BASE}}$. A curve with a larger area under it indicates a better accuracy–efficiency trade-off.

We can see that the blue curves are basically linear interpolations between the LSTM (the lower-left dot) and $\text{BERT}_{\text{BASE}}$ (the upper-right dot), and this is expected for random selection. Orange and green curves are concave, indicating that using SR for confidence estimation is, un-

[30]The cost of $\text{BERT}_{\text{BASE}}$ is $\sim 10^5$ times larger than LSTM here.

[31]We use the torchprofile toolkit to measure multiply–accumulate operations (MACs), and then double the number to obtain floating point operations (FLOPs).

surprisingly, more effective than random selection. Between these two, the green curves (history error regularization) have larger areas under themselves than orange ones (no regularization), i.e., green curves have better accuracy given the same FLOPs. This demonstrates the effectiveness of error regularization for better confidence estimation.

6.6 Summary

In this chapter, we discuss the problem of selective prediction for NLP. We provide theoretical background and evaluation metrics for the problem, and also propose a simple error regularization method that improves selective prediction performance for NLP models. We conduct experiments to compare different models under the selective prediction setting, demonstrate the effectiveness of the proposed regularization trick, and study two scenarios where selective prediction and the error regularization method can be helpful.

We summarize interesting experimental observations as follows:

1. Recent sophisticated NLP models not only improve accuracy over simple models, but also provide better selective prediction results (better confidence estimation).

2. MC-dropout, despite having a solid theoretical foundation, has difficulties matching the effectiveness of simple SR in practice.

3. The simple error regularization helps models lower their AUC and RPP, i.e., models trained with it produce better confidence estimators.

4. Selective prediction can be applied to scenarios where estimating example difficulties is necessary. In these cases, our proposed error regularization trick can also be helpful, such as providing better accuracy–efficiency trade-offs.

Chapter 7

Commutativity and Cumulativeness of Efficiency Operators

With numerous efficiency methods available, including early exiting and others such as distillation, pruning, quantization, etc., it is natural to consider applying multiple methods sequentially to a model. In this chapter, we study how to construct a pipeline of efficiency methods and apply them to transformer models.

7.1 Overview

Besides early exiting, a wide variety of efficiency methods have been *individually* studied for transformers, like pruning (McCarley et al., 2019), distillation (Sanh et al., 2019), dynamic sequence length (Kim and Cho, 2020), and quantization (Shen et al., 2020), just to name a few. With all the individual efficiency methods available, there has also been work on combining multiple ones, including reducing multiple dimensions of a model simultaneously and sequentially as a pipeline. For example, DynaBERT (Hou et al., 2020) improves model efficiency by first reducing model width and then reducing depth. Cui et al. (2021) perform pruning and distillation jointly for model compression. Lin et al. (2021b) propose a bag of tricks to accelerate the inference stage of neural machine translation models. Fastformers (Kim and Hassan, 2020) propose a

pipeline consisting of several components which together provide more than $100\times$ acceleration. Despite the success of constructing an efficiency pipeline, *how* these pipelines should be built in order to achieve the best accuracy–efficiency trade-offs has not been methodically studied. Firstly, we do not know how to choose components for the pipeline among numerous options, since time savings often come at the price of accuracy drops, and therefore, naïvely stacking all available efficiency methods often leads to poor performance. Secondly, even with a chosen set of efficiency methods, it is unclear whether we need to exhaustively examine all possible orders to find the best one.

In this chapter, we study how to effectively construct a pipeline of efficiency methods. Among the three stages of applying transformers, pre-training, fine-tuning, and inference, we assume the availability of pre-trained models and study different ways of fine-tuning them to achieve better trade-offs between inference accuracy and efficiency.[32] Conceptually, we consider each efficiency method as an *operator* applied on a model and study the properties of these efficiency operators. We conduct experiments with the RoBERTa model (Liu et al., 2019) on a number of NLP tasks and include the following components in our efficiency pipelines: distillation, structured pruning, quantization, early exiting, and dynamic length inference. We study two important properties of efficiency operators: (1) Commutativity: does arbitrarily swapping the order of operators affect the final accuracy–efficiency trade-off of the model? (2) Cumulativeness: how do the two metrics, time savings and accuracy drops, compound across multiple operators? For commutativity, we show that the difference between various orderings of the same set of components is usually small and negligible in practice. For cumulativeness, we show that time saving and accuracy drop are both cumulative to the extent that we can estimate the performance of a new pipeline by combining the results of individual components. These observations make it more convenient for us to build new pipelines and estimate their performance without having to carry out time-consuming experiments.

7.2 Modeling Details for Operators

In this section, we introduce modeling details for operators that we use in this chapter.

[32] *Training* henceforth refers to fine-tuning in the chapter.

7.2.1 Knowledge Distillation

Knowledge distillation (Hinton et al., 2015) improves efficiency by distilling knowledge from a large and costly *teacher* model to a small and efficient *student* model. In the case of transformers, there are two types of distillation, namely task-agnostic and task-specific, depending on whether the student model is trained for a specific task.

In this chapter, we focus on *task-specific distillation*, which corresponds to the fine-tuning stage. We initialize the student model with a TinyBERT (Jiao et al., 2020) backbone that comes from task-agnostic distillation. In addition to the most common loss function (teacher supervising student), which is a soft cross-entropy between output logits of the teacher and the student, we introduce two other parts for the loss function: (1) mean squared error (MSE) between the teacher's and the student's embedding layers' outputs; (2) MSE between the teacher's and the student's final transformer layers' outputs. It has been shown in related work that adding objectives to align intermediate states of the teacher and the student helps with distillation (Sun et al., 2019; Sanh et al., 2019). We simply use a ratio of $1:1:1$ for these three parts of the loss function.

7.2.2 Structured Pruning

Pruning removes unimportant parts of the model and increases the sparsity level of the model. A specific category of pruning, *structured pruning* (Han et al., 2015; Anwar et al., 2017; Gordon et al., 2020), removes high-level units of the model, such as a layer, an attention head, or an entire row/column in an FFN's weight matrix. Model sparsity induced by structured pruning can directly translate to faster execution, and therefore we focus on structured pruning in the chapter.

In this chapter, we follow the work by McCarley et al. (2019); Kim and Hassan (2020) and choose two aspects of the model and prune them separately: the number of attention heads and the intermediate dimension of the FFN layer within a transformer layer. We calculate the *importance* of attention heads and intermediate dimensions with a first-order method: run inference for the entire dev set and accumulate the first-order gradients for each attention head and intermediate dimension. We then remove the least important attention heads and intermediate dimensions, according to the desired sparsity level, and then rewire the model connection so it becomes a smaller but complete model. After pruning, we perform another round of knowledge distillation

from the original model to the pruned model as described in the previous subsection, which further improves the pruned model's accuracy without sacrificing efficiency.

7.2.3 Dynamic Inference

Dynamic inference (Teerapittayanon et al., 2016; Graves, 2016b; Dehghani et al., 2019) accelerates inference by reducing the amount of computation adaptively, depending on the nature of the input example. We discuss two types of dynamic inference.

Early Exiting

Early exiting, described in Chapters 3 to 5, is a dynamic depth inference method. For training, we simply use the *Joint* fine-tuning method discussed in Chapter 4, since we mostly work with non-low-resource datasets, where *Joint* provides good performance with straightforward implementation and fast training. We also use confidence as the early exiting criterion and inference is the same as described in Chapter 4.

A special case to notice here is that the training of distillation and pruning needs to be adjusted *after* adding early exiting.

- Distillation after early exiting. When we initialize the student model (e.g., from TinyBERT), we also add early exiting classifiers to it. For training, the ith layer of the student model uses the prediction from the $2i^{\text{th}}$ layer of the teacher model as supervision.
- Pruning after early exiting. When we prune the transformer layers, we do not change the classifiers. For the additional round of distillation, each layer of the student model uses the prediction from its corresponding layer of the teacher model as supervision.

Dynamic Sequence Length

We use a simple method for length reduction: for each batch, we dynamically set the input sequence length to the maximum length of inputs within the batch. This reduces the number of

zero paddings in input sequences and reduces unnecessary computation. Different from previous methods, dynamic sequence length does not affect the model's accuracy.

7.2.4 Quantization

Quantization (Lin et al., 2016; Shen et al., 2020) improves model efficiency by using fewer bits to store and process data. The idea itself is straightforward, but implementation can be highly hardware dependent. Since we run inference on CPUs, we first export the trained model to ONNX[33] and then run it with 8-bit quantization, following Fastformers (Kim and Hassan, 2020).

7.3 Experimental Design

In this section, we introduce the detailed design and setups for our experiments.

7.3.1 Conceptual Framework

In our experiments, we work with *pipelines* consisting of multiple efficiency *operators*. We represent a pipeline with a string of bald capital letters, where each letter represents an efficiency operator and the order of these letters represents their order.

The operators include: **D**istillation, Structured **P**runing, **E**arly Exiting, Dynamic **L**ength, and **Q**uantization. For example, the string "**DEPLQ**" represents a pipeline of sequentially applying the following operators to a fine-tuned model: (1) distill it into a student model; (2) add early exiting classifiers to it and train; (3) apply structured pruning to make each layer "thinner" and distill from the unpruned model; and (4) use dynamic length and quantization for the final inference. Additionally, we use **O** to represent an "empty" pipeline, i.e., directly applying the **O**riginal fine-tuned model.

Not all combinations of operators constitute a meaningful pipeline. Among the operators discussed in this chapter, **D**, **P**, and **E** require additional training steps, while **Q** and **L** are directly

[33] https://onnx.ai/.

applicable right before inference. Therefore, **D**, **P**, and **E** (Group I) should always appear before **Q** and **L** (Group II) in the pipeline. Moreover, applying **D** after **P** does not make sense, since **D** initializes a small student, and the efficiency brought by the pruning step cannot be passed over to the student. With these constraints, the number of meaningful pipelines is significantly reduced.

7.3.2 Datasets and Implementation

We conduct experiments with the RoBERTa$_{\text{BASE}}$ model (Liu et al., 2019) on four sequence classification tasks: MRPC, SST-2, QNLI, and QQP (Section 2.3). Our implementation of efficiency methods are adopted from Fastformers (Kim and Hassan, 2020) and DeeBERT (Chapter 3). We train all the models with an NVIDIA Tesla T4 GPU. We evaluate them with an AMD Ryzen 5800X CPU, where we measure the wall-clock time for inference.

7.3.3 Settings for Pipelines and Operators

Before experimenting with pipelines, we explore the optimal setting (e.g., learning rate, batch size) for each individual operator and use the same setting in the pipelines. This is a realistic approach since it is impractical to search for the optimal setting for every component in every new pipeline.

For training the RoBERTa model, including original fine-tuning, distillation, and training with early exiting, we use the same hyperparameters as in the Transformers library (Wolf et al., 2020): learning rate is set to 10^{-5}; batch size is set to 8; all training procedures consist of 10 epochs with no early stopping. For pruning, we prune the number of attention heads from 12 to 8 and the intermediate dimension from 3072 to 1536—in our preliminary experiments, this combination is a sweet spot on the Pareto frontier.

7.4 Operator Commutativity and Order

Given a set of operators, we naturally wonder about the best order to apply them. Although this question seems formidable due to the exponentially large number of possible orderings, we

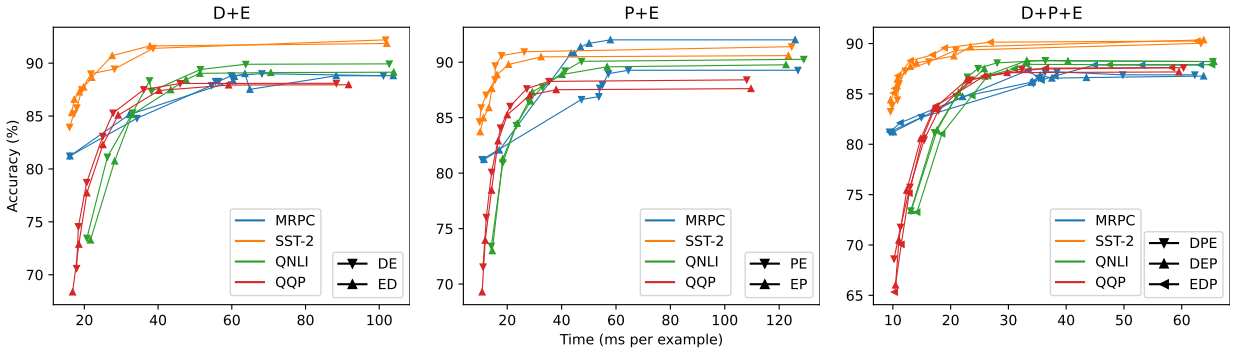


Figure 7.1: Different orderings of the same set of operators have similar trade-off curves. The title of each subfigure shows the set of operators; each color represents a dataset; each marker shape represents a component ordering.

show that the question is actually simpler than expected: on the one hand, we have eliminated a number of invalid orderings as described in Section 7.3; on the other, we show that operators are commutative in the remaining ordering candidates.

7.4.1 Commutative Properties of Operators

In this subsection, we show how commutative these operators are, i.e., how much difference swapping their orders makes. We discuss the two groups separately.

Group I We show the results of swapping the order of operators from Group I in Figure 7.1. Since early exiting is involved, which means the model can achieve different trade-offs between accuracy and inference time, we present each ordering as a *trade-off curve*, where points are drawn by varying the early exiting threshold of confidence. We can see that when we use the same set of operators, different orderings have similar trade-off curves, in most cases.

Exceptions exist, however, in the **E**+**P** combination on the MRPC dataset. We hypothesize that this is due to training randomness, since MRPC has smallest size of all. In order to study randomness, we repeat the experiment with additional random seeds and show in Figure 7.2 the results on MRPC. We can see that (1) the gap between the *mean* curves is smaller than the

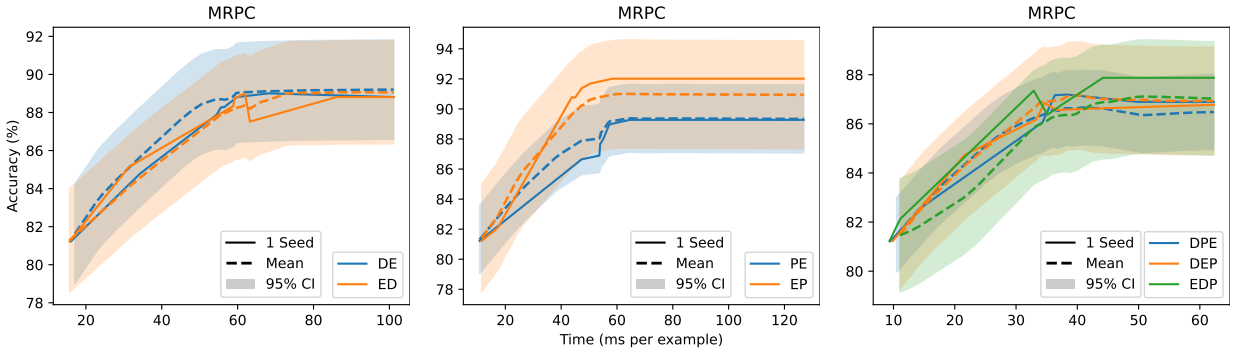


Figure 7.2: Comparing the results of a single run (solid lines; same as the ones from Figure 7.1) and the results from multiple runs (dashed lines for the mean and shaded areas for 95% confidence intervals).

gap between curves corresponding to using a single seed; (2) the mean curve of each ordering lies within the 95% confidence interval (95% CI) of other orderings. This shows that the differences between tradeoff curves of different orderings can, at least partly, be attributed to training randomness.

To further quantify the degree of dissimilarity between different orderings, we define and calculate the *distance* between tradeoff curves. The distance between two tradeoff curves is defined as the maximum accuracy (y-axis) difference at the same inference time (x-axis) point. We compare distances between tradeoff curves (1) generated by the same operator order but with different random seeds; and (2) generated by different operator orders. We show the results in Table 7.1. We can see that while tradeoff curves generated by the same operator order tend to have a smaller average distance, the difference between same/different orders is typically small and the one-standard-deviation (1-SD) intervals of both sides always overlap. Although we are unable to find a suitable significance test since the distances are not independent, the above analysis shows that the difference of distances between curves from same/different orders is likely not significant. More importantly, as shown in Figure 7.2, the gap between mean curves of different orderings is smaller than the deviation caused by different random seeds. Therefore, in practice, we can regard the operators as commutative.

Dataset	Order	D+E	P+E	D+P+E
MRPC	Same	1.57 ± 0.69	2.31 ± 0.85	1.53 ± 0.62
	Diff.	1.74 ± 0.40	4.12 ± 1.10	2.59 ± 0.97
SST-2	Same	1.30 ± 0.39	1.64 ± 0.46	1.49 ± 0.53
	Diff.	1.48 ± 0.46	1.84 ± 0.62	1.98 ± 0.82
QNLI	Same	2.24 ± 1.20	4.40 ± 2.49	3.41 ± 2.51
	Diff.	3.93 ± 0.82	4.58 ± 2.39	4.91 ± 2.44
QQP	Same	2.38 ± 1.36	2.11 ± 0.86	2.30 ± 1.05
	Diff.	3.64 ± 1.14	3.30 ± 1.27	4.56 ± 1.71

Table 7.1: The mean and the standard deviation (SD) of *distances* between trade-off curves belonging to same/different orders (the same ordering is run with multiple random seeds). For all entries, the 1-SD intervals of same/different orders overlap.

Group II The two operators, **Q** and **L**, are independent of each other, and therefore their order can be arbitrarily swapped (i.e., they are strictly commutative by definition). We show the results of applying **Q** and/or **L** at the end of different pipelines in Table 7.2. We do not report the accuracy of +**L** since using dynamic length does not change the model's accuracy.

Based on the above discussion, when we have a set of components to apply, it suffices to simply pick a reasonable order from the candidate space.

7.5 Operator Cumulativeness and Predictability of Pipelines

In order to choose components for an efficiency pipeline, an important question is whether time savings and accuracy drops of individual operators are cumulative. In this subsection, we show that they are indeed cumulative to the degree that accuracy–efficiency trade-offs of a new pipeline can be estimated by combining the results of individual operators.

Dataset	Pipeline	Accuracy (%)		Time (ms per example)				
		Raw	**+Q** (relative diff.)	Raw	**+Q**	**+L**	**+QL**	**+QL** (est.)
MRPC	**O**	92.7	92.5 (−0.2%)	170.7	−50%	−83%	−94%	−92%
	D	89.2	88.8 (−0.4%)	85.5	−49%	−82%	−94%	−91%
	P	91.0	89.0 (−2.2%)	122.4	−64%	−86%	−94%	−95%
	DP	88.9	87.9 (−1.1%)	59.3	−62%	−84%	−94%	−94%
SST-2	**O**	93.7	93.5 (−0.2%)	170.8	−50%	−86%	−97%	−93%
	D	92.3	92.3 (−0.0%)	85.5	−49%	−86%	−97%	−93%
	P	92.4	91.7 (−0.8%)	126.7	−66%	−89%	−97%	−96%
	DP	92.0	90.9 (−1.2%)	62.9	−65%	−88%	−97%	−96%
QNLI	**O**	92.3	92.1 (−0.2%)	174.2	−51%	−83%	−95%	−92%
	D	91.3	90.7 (−0.7%)	86.9	−50%	−82%	−95%	−91%
	P	91.5	91.4 (−0.1%)	121.5	−64%	−86%	−95%	−95%
	DP	89.8	89.6 (−0.2%)	62.6	−65%	−85%	−95%	−95%
QQP	**O**	88.6	88.3 (−0.3%)	172.3	−51%	−86%	−96%	−93%
	D	87.9	87.7 (−0.2%)	88.2	−51%	−85%	−97%	−93%
	P	88.5	88.5 (−0.0%)	118.3	−63%	−87%	−97%	−95%
	DP	87.6	87.6 (−0.0%)	58.8	−62%	−86%	−97%	−95%

Table 7.2: Accuracy drops and time savings provided by quantization (**Q**) and dynamic length inference (**L**) applied at the end of pipelines. The accuracy drops and time savings of most operators are cumulative.

We first discuss operators from Group I. In Figure 7.3, we show how we can estimate the trade-off curve of a new pipeline based on the results of its constituents, using the two larger and more stable datasets, QQP and QNLI. For example, in the top-right subfigure, we show the estimation for the trade-off curves of pipelines comprising **E**, **D**, and **P**, based on the results of individually applying each of these operators.

The idea for estimating accuracy drops is based on the following *cumulativeness assumption*.

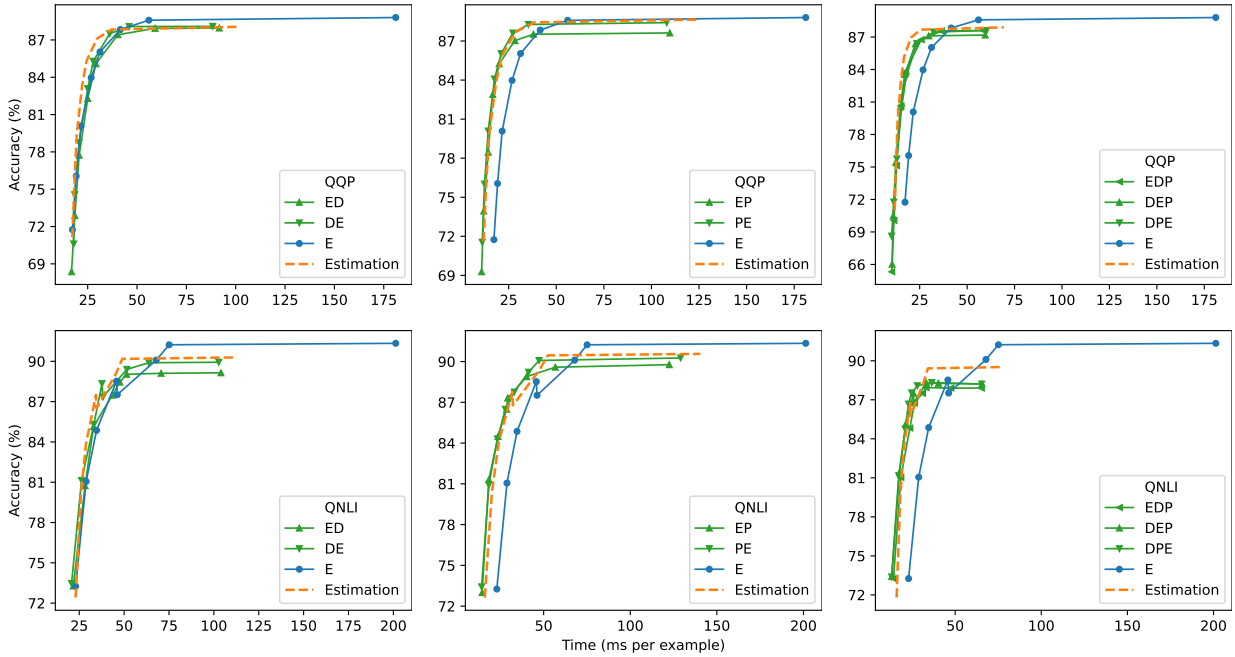


Figure 7.3: Estimating the trade-off curves of target pipelines based on the results of individually applying operators. Green curves: measured trade-off curves of target pipelines; blue curves: measured trade-off curves of individually applying the operator **E**; orange curves: estimated trade-off curves for the target pipelines.

Suppose **R** is a pipeline and A_* is the accuracy for a pipeline *, the assumption is:

$$A_{\mathbf{R+D}} = \frac{A_{\mathbf{D}}}{A_{\mathbf{O}}} \times A_{\mathbf{R}}, \tag{7.1}$$

$$A_{\mathbf{R+P}} = \frac{A_{\mathbf{P}}}{A_{\mathbf{O}}} \times A_{\mathbf{R}}. \tag{7.2}$$

In other words, our assumption is that adding **D** or **P** to *any* pipeline should result in similar relative accuracy drops. We can therefore estimate the accuracy of **ED**, **EP**, and **EDP** (and other orders of the same set of operators) as follows: (1) calculate accuracy drops of **D** and **P** relative to **O**; (2) multiply the relative accuracy drops to points on **E**'s trade-off curve.

The idea for estimating time savings is also similar, but additional modifications are necessary:

- When we add **P** to **E**, since they work on reducing different dimensions of the model (width and depth), the time savings are independent and directly cumulative:

$$T_{\mathbf{E+P}} = \frac{T_{\mathbf{P}}}{T_{\mathbf{O}}} \times T_{\mathbf{E}}, \tag{7.3}$$

  where similarly, T_* is the inference time for a pipeline *.

- When we add **D** to **E**, we need to consider the fact that both **D** and **E** reduce the number of layers. When the early exiting threshold is extremely large and the model uses all layers for inference, the relative time saving will be close to $T_{\mathbf{D}}/T_{\mathbf{O}}$; when the early exiting threshold is extremely small and the model exits after only one layer, adding **D** provides no extra time saving. Therefore, for non-extreme cases, we can estimate the time saving for **E+D** by interpolating the above two extreme cases:

$$T_{\mathbf{E+D}} = t_{\mathbf{E}} + (T_{\mathbf{E}} - t_{\mathbf{E}}) \times \frac{T_{\mathbf{D}}}{T_{\mathbf{O}}}, \tag{7.4}$$

  where $t_{\mathbf{E}}$ is the minimum value of time in the trade-off curve of **E** (i.e., the point where we early exit after only one layer).

- When we add both **P** and **D** to **E**, we combine the above two estimations:

$$T_{\mathbf{E+DP}} = \left(t_{\mathbf{E}} + (T_{\mathbf{E}} - t_{\mathbf{E}}) \times \frac{T_{\mathbf{D}}}{T_{\mathbf{O}}}\right) \times \frac{T_{\mathbf{P}}}{T_{\mathbf{O}}}. \tag{7.5}$$

We use the above ideas to estimate trade-off curves of new pipelines and show the results in Figure 7.3. From the figure, we can see that the estimation curves (orange) align well with the measured curves (green), across different datasets and operator sets, showing that individual components from Group I are cumulative with each other under these settings.

For operators from Group II, we refer to Table 7.2. We see that on the same dataset, **Q** leads to similar accuracy drops when added to any pipeline, especially on the larger and more stable datasets, QNLI and QQP. Time savings, on the other hand, are trickier:

- **L** provides consistent time savings for all pipelines, showing that it is cumulative with any operator from Group I.

- **L** and **Q** are also cumulative with each other, as evidenced by the fact that the measured time savings of **+QL** align well with the estimation of **+QL**, which is simply multiplying the respective savings of **Q** and **L**.

- **Q**, however, is cumulative only with **D** and **E**, but not **P**—it saves more time for pipelines with **P**. This is because quantization's acceleration is different for different types of operations, and pruning changes the proportion of each type of operations within a transformer layer, while distillation or early exiting does not. When we estimate the trade-off of a pipeline containing both **Q** and **P**, **PQ** needs to be treated as a compound operator, and it is cumulative with others. This also applies to other operators that change the connection within a transformer layer.

Empirically, the observation that operators are cumulative facilitates future experiments on efficiency pipelines: for pipelines that are tedious to train and evaluate, simply measuring the performance of their components can provide us with a reliable estimation of the pipeline's behavior. Theoretically, the observation also makes it easier to analyze the contribution of each component to the pipeline (i.e., how much time does each one save and how much accuracy does each one sacrifice): the Shapley value (Shapley, 1997) of each component can be approximated by simply using the standalone estimation (Fréchette et al., 2016).

7.6 Summary

In this chapter, we consider efficiency methods as operators applied on transformer models and study the properties of these operators. We observe from experiments that (1) operators are commutative: changing their order has little practical impact on the final efficiency–accuracy trade-off; (2) operators are cumulative: a new pipeline's performance can be estimated by cumulating time savings and accuracy drops of each component. These observations facilitate the future construction of efficiency pipelines and also provide an interesting direction to better understand efficiency pipelines.

Chapter 8

Conclusion and Future Work

The transformer is a powerful model that is widely used in NLP and IR applications, yet its efficient inference has been a challenging problem for the community. Considering its multi-layer architecture where the layers share the similar structure, we apply early exiting to reduce the effective depth of the model and thereby improve efficiency.

We make the initial attempt on sequence classification tasks on the GLUE benchmark (Wang et al., 2018a), and experiments show that we can reduce inference runtime by as much as $40 \sim 50\%$. Later on, we discuss a few extensions of the combination of early exiting and transformers. First, we improve the fine-tuning strategy of the multi-output model to combine the benefits of several existing fine-tuning strategies and manage to achieve a better balance between layers, which leads to better accuracy–efficiency trade-offs. Second, we extend the idea from classification-only to other tasks including regression, by introducing an auxiliary learning-to-exit module that explicitly learns to make the exiting decision. Third, we apply early exiting for document reranking, proposing asymmetric early exiting for this asymmetric problem and improving efficiency by more than $2\times$.

Extrapolating the early exiting idea, we consider selective prediction in the case of transformers, where we abstain from making predictions when confidence is low. We propose a regularization trick that can improve confidence estimation of the model. We also explore two interesting scenarios where selective prediction is useful and our proposed regularization provides better performance.

We also consider the idea of combining early exiting with other efficiency methods such as distillation, structured pruning, quantization, dynamic sequence length, etc. We regard each of these efficiency methods as an operator applied on a transformer model. Experiments show that these operators are commutative and cumulative, i.e., the order of applying them has little impact on the final accuracy–efficiency trade-off, and the final trade-off can be estimated by combining the accuracy drop and efficiency gain of each component. These discoveries have both empirical and theoretical meanings.

At the end of the thesis, we discuss a number of interesting future directions for further research on this topic.

- **Batch inference**. As mentioned in Section 5.4, our current implementation of early exiting, including sequence classification, sequence regression, document reranking, etc., only supports the inference of single examples. This is due to the dynamic nature of early exiting: in batch inference, the confidence is different for each example within the batch. However, in real-world applications, doing inference in batches is not uncommon, especially when the amount of inference requests is high. To enable batch inference, we can calculate the confidence of each example at each layer and perform early exiting as follows. For examples whose confidence is high enough, we terminate inference and record the results; for the rest, we *reorganize* the batch and continue with the next layer's inference. In the end, we obtain the final results by combining results recorded at different layers. This main challenge is on the implementation side and may not be of significant research interest. However, it will certainly be very meaningful for industrial applications.

- **Early exiting for token-level prediction**. Similar to batch inference, early exiting for token-level prediction, such as span prediction and named entity recognition, faces a similar difficulty: there exist multiple values of confidence at a certain layer and it is not clear how to make the exiting decision. We can also take a similar approach: make the prediction for each token at each layer, early exiting the confident ones, and continue with the rest. At the final layer, we recover the result for the entire sequence. However, there are two obvious issues with this simple approach: (1) When we early exit some tokens at a certain layers, it is unclear how to compute the multi-head attention for other tokens at future layers; (2) The overhead may be too high since each token requires the computation of

the classifier. Nevertheless, the problem itself is important (especially for auto-regressive transformer models) and the approach can serve as a competitive baseline to facilitate further improvements.

- **Early exiting for sequence-to-sequence tasks**. Sequence-to-sequence (seq2seq) (Sutskever et al., 2014; Bahdanau et al., 2014) is an important architecture used in a wide variety of NLP and IR applications such as machine translation, text summarization, query expansion, etc. Different from models such as BERT and RoBERTa, seq2seq models typically have two components, encoder and decoder, both of which can be transformers (Raffel et al., 2020; Lewis et al., 2020a). We can, of course, extend the idea of early exiting to seq2seq models. In this case, a new challenge arises: there are a lot of places where we can perform early exiting and it is unclear what choices are optimal. For example, we can early exit the encoder, the decoder, or both. For both the encoder and the decoder, we can perform token-level or sequence-level early exiting. The problem itself is complex and interesting, and worth thorough exploration.

- **Better confidence estimation for early exiting**. The performance of early exiting large depends on confidence estimation. With accurate confidence estimation, the model can allocate computation to examples that need more layers, thereby achieving good accuracy–efficiency trade-offs. In Chapter 6, we discuss confidence estimation and propose a regularization method to improve it. We show that softmax response, which is already used in Chapter 4 and also other papers (Schwartz et al., 2020), is lightweight yet powerful. On the other hand, our followup experiments show that the proposed regularization method, while being useful in scenarios such as the no-answer problem and classifier cascades, is not enough to provide significant improvements to early exiting. Perhaps confidence estimation for deep learning models is inherently difficult, but if we can significantly improve it, that will certainly bring early exiting to a new stage.

- **A representative demonstration application**. Most of our experiments are done in a research setting without an intuitive visualization, and it will be interesting to implement a concrete demonstration application to show the improvements in efficiency in a straightforward way. For example, we can design a two-stage search engine, where the first stage is dense retrieval (Karpukhin et al., 2020) and the second stage is reranking (Nogueira and

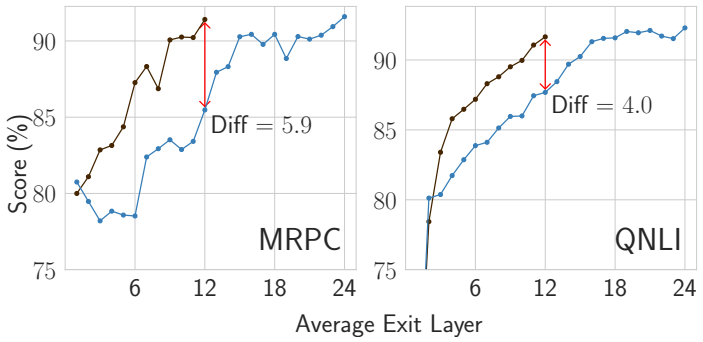


Figure 8.1: Comparison between the *limit* scores of $\text{BERT}_{\text{BASE}}$ (brown) and $\text{BERT}_{\text{LARGE}}$ (blue). The y-axis shows the score (F1 for MRPC and accuracy for QNLI) relative to vanilla fine-tuning of $\text{BERT}_{\text{LARGE}}$. The red arrow shows the difference between the two models when they both use the same number of layers (12).

Cho, 2019). When run on the mobile browser, it takes a relatively long time to search for a query. But after adding the methods proposed in this thesis, the search time can hopefully be reduced to a fraction, greatly improving the user's experience.

- **Early exiting in pre-training**. When designing modeling details for early exiting (how to fine-tune, how to make exiting decisions, etc.), the fundamental question is how many transformer layers are necessary for making good predictions. We use a small experiment to show that the answer to this question largely depends on how the model is pre-trained. The experiment is called *the upper limit of early exiting*. We take the first k transformer layers from a pre-trained transformer model, attach a classifier to the end of this k-layer model, and fine-tune this single-output model. Our previous experiments have shown that different classifiers typically interfere with each other negatively and it is hard to achieve the optimal accuracy for all of them. This experiment estimates the upper bound for each layer by removing inter-classifier interference. We compare the *limit* performance of $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$ in Figure 8.1 on MRPC (Dolan and Brockett, 2005) and QNLI (Rajpurkar et al., 2016; Wang et al., 2018a). We notice that with the same number of layers, $\text{BERT}_{\text{BASE}}$ almost always outperforms $\text{BERT}_{\text{LARGE}}$ by a large margin. This show that the final layers of $\text{BERT}_{\text{BASE}}$ are much more "prepared" for early exiting than the

intermediate layers of $\text{BERT}_{\text{LARGE}}$. The gap suggests that most transformer layers' potential to provide information for early exiting prediction is limited by the single-output nature of pre-training: the intermediate layers are solely optimized to provide inputs to the following layers. Therefore, if we want to further improve early exiting for better accuracy at earlier layers, adding more exiting paths in pre-training will be a promising direction. Perhaps, for efficiency methods in general (especially pruning ones), working directly on pre-training will be more fruitful than just fine-tuning, as exemplified by the comparison between task-agnostic and task-specific distillation.

References

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. 2016. MS MARCO: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. In *TAC*.

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, pages 1877–1901. Curran Associates, Inc.

B. Barla Cambazoglu, Hugo Zaragoza, Olivier Chapelle, Jiang Chen, Ciya Liao, Zhaohui Zheng, and Jon Degenhardt. 2010. Early exit optimizations for additive machine learned ranking systems. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 411–420.

Mark Carlebach, Ria Cheruvu, Brandon Walker, Cesar Ilharco Magalhaes, and Sylvain Jaume. 2020. News aggregation with diverse viewpoint identification using neural embeddings and semantic understanding models. In *Proceedings of the 7th Workshop on Argument Mining*, pages 59–66, Online. Association for Computational Linguistics.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Eunsol Choi, Omer Levy, Yejin Choi, and Luke Zettlemoyer. 2018. Ultra-fine entity typing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 87–96, Melbourne, Australia. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.

David Cohn, Zoubin Ghahramani, and Michael Jordan. 1995. Active learning with statistical models. In *Advances in Neural Information Processing Systems*, volume 7. MIT Press.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2020. Overview of the TREC 2019 deep learning track. *arXiv preprint arXiv:2003.07820*.

Baiyun Cui, Yingming Li, and Zhongfei Zhang. 2021. Joint structured pruning and dense knowledge distillation for efficient transformer model compression. *Neurocomputing*, 458:56–69.

Andrew M. Dai and Quoc V. Le. 2015. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le. 2020. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. In *Advances in Neural Information Processing Systems*, volume 33, pages 4271–4282. Curran Associates, Inc.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *International Conference on Learning Representations*.

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, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Ran El-Yaniv and Yair Wiener. 2010. On the foundations of noise-free selective classification. *Journal of Machine Learning Research*, 11(53).

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive transformer. In *International Conference on Learning Representations*.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*.

Alexandre Fréchette, Lars Kotthoff, Tomasz Michalak, Talal Rahwan, Holger Hoos, and Kevin Leyton-Brown. 2016. Using the Shapley value to analyze algorithm portfolios. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA. PMLR.

Siddhant Garg, Thuy Vu, and Alessandro Moschitti. 2019. Tanda: Transfer and adapt pre-trained transformer models for answer sentence selection. *arXiv preprint arXiv:1911.04118*.

Yonatan Geifman and Ran El-Yaniv. 2017. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

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*, pages 143–155, Online. Association for Computational Linguistics.

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 37th International Conference on*

Machine Learning, volume 119 of *Proceedings of Machine Learning Research*, pages 3690–3699. PMLR.

Alex Graves. 2016a. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Alex Graves. 2016b. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*.

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

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Sebastian Hofstätter and Allan Hanbury. 2019. Let's measure run time! extending the ir replicability infrastructure to include performance aspects. *arXiv preprint arXiv:1907.04614*.

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, pages 9782–9793. Curran Associates, Inc.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. 2018. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*.

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 *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Amita Kamath, Robin Jia, and Percy Liang. 2020. Selective question answering under domain shift. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5684–5696, Online. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Gyuwan Kim and Kyunghyun Cho. 2020. Length-adaptive transformer: Train once with length drop, use anytime with search. *arXiv preprint arXiv:2010.07003*.

Young Jin Kim and Hany Hassan. 2020. FastFormers: Highly efficient transformer models for natural language understanding. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 149–158, Online. Association for Computational Linguistics.

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 *International Conference on Learning Representations*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020a. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020b. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2849–2858, New York, New York, USA. PMLR.

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. 2017. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021a. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2356–2362. ACM.

Ye Lin, Yanyang Li, Tong Xiao, and Jingbo Zhu. 2021b. Bag of tricks for optimizing transformer efficiency. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4227–4233, Punta Cana, Dominican Republic. Association for Computational Linguistics.

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*, pages 6035–6044, Online. Association for Computational Linguistics.

Yijin Liu, Fandong Meng, Jie Zhou, Yufeng Chen, and Jinan Xu. 2021. Faster depth-adaptive transformers. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(15).

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

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland. European Language Resources Association (ELRA).

JS McCarley, Rishav Chakravarti, and Avirup Sil. 2019. Structured pruning of a bert-based question answering model. *arXiv preprint arXiv:1910.06360*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Jooyoung Moon, Jihyo Kim, Younghak Shin, and Sangheum Hwang. 2020. Confidence-aware learning for deep neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7034–7044. PMLR.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with BERT. *arXiv preprint arXiv:1901.04085*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

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, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Mary Phuong and Christoph H. Lampert. 2019. Distillation-based training for multi-exit architectures. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Fernando J. Pineda. 1987. Generalization of back-propagation to recurrent neural networks. *Physical review letters*, 59(19):2229.

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

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(140):1–67.

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*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Stephen E. Robertson and Karen Spärck Jones. 1976. Relevance weighting of search terms. *JASIS*, 27(3):129–146.

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

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. 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, Online. Association for Computational Linguistics.

Lloyd S Shapley. 1997. A value for n-person games. *Classics in game theory*, 69.

Lakshay Sharma, Laura Graesser, Nikita Nangia, and Utku Evci. 2019. Natural language understanding with the Quora Question Pairs dataset. *arXiv preprint arXiv:1907.01041*.

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05).

Yanyao Shen, Hyokun Yun, Zachary C. Lipton, Yakov Kronrod, and Animashree Anandkumar. 2018. Deep active learning for named entity recognition. In *International Conference on Learning Representations*.

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

Luca Soldaini and Alessandro Moschitti. 2020. The cascade transformer: an application for efficient answer sentence selection. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5697–5708, Online. Association for Computational Linguistics.

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)*, pages 4323–4332, Hong Kong, China. Association for Computational Linguistics.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. 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, Online. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. 2019. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*.

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 *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018a. 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, Brussels, Belgium. Association for Computational Linguistics.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020. HAT: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7675–7688, Online. Association for Computational Linguistics.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. 2018b. SkipNet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. 2018. BlockDrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ji Xin, Raphael Tang, Zhiying Jiang, Yaoliang Yu, and Jimmy Lin. 2022. Building an efficiency pipeline: Commutativity and cumulativeness of efficiency operators for transformers. *arXiv preprint arXiv:2208.00483*.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020a. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online. Association for Computational Linguistics.

Ji Xin, Rodrigo Nogueira, Yaoliang Yu, and Jimmy Lin. 2020b. Early exiting BERT for efficient document ranking. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 83–88, Online. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. 2021a. BERxiT: Early exiting for BERT with better fine-tuning and extension to regression. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 91–104, Online. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. 2021b. The art of abstention: Selective prediction and error regularization for natural language processing. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1040–1051, Online. Association for Computational Linguistics.

Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the use of Lucene for information retrieval research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1253–1256. ACM.

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. Curran Associates, Inc.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2019. Slimmable neural networks. In *International Conference on Learning Representations*.

Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Computer Vision – ECCV 2014*, pages 818–833, Cham. Springer International Publishing.

Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. 2019a. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. TernaryBERT: Distillation-aware ultra-low bit BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 509–521, Online. Association for Computational Linguistics.

Xuchao Zhang, Fanglan Chen, Chang-Tien Lu, and Naren Ramakrishnan. 2019b. Mitigating uncertainty in document classification. 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 3126–3136, Minneapolis, Minnesota. Association for Computational Linguistics.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. BERT loses patience: Fast and robust inference with early exit. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18330–18341. Curran Associates, Inc.