UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Efficient Natural Language Processing for Language Models

Permalink

https://escholarship.org/uc/item/9dv1k5xv

Author

Xu, Canwen

Publication Date

2024

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA SAN DIEGO

Efficient Natural Language Processing for Language Models

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Canwen Xu

Committee in charge:

Professor Julian McAuley, Chair
Professor Taylor Berg-Kirkpatrick
Professor Zhiting Hu
Professor Jingbo Shang

2024

Copyright

Canwen Xu, 2024

All rights reserved.

The Dissertation of Canwen Xu is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2024

TABLE OF CONTENTS

Dissertation Approval Page iii

Table of Contents iv

List of Figures vii

List of Tables ix

Acknowledgements xi

Vita xiv

Abstract of the Dissertation xv

Chapter 1 Introduction 1
  1.1 Computation Efficiency 2
  1.2 Data Efficiency 3
  1.3 Summary 3

Chapter 2 Background 5
  2.1 Language Models 5
  2.2 Measuring Efficiency of Language Models 6
  2.3 Conclusion 8

Chapter 3 BERT Loses Patience: Fast and Robust Inference with Early Exit 9
  3.1 Introduction 9
  3.2 Patience-based Early Exit 11
    3.2.1 Motivation 11
    3.2.2 Inference 12
    3.2.3 Training 14
    3.2.4 Theoretical Analysis 14
  3.3 Experiments 15
    3.3.1 Tasks and Datasets 15
    3.3.2 Baselines 16
    3.3.3 Experimental Setting 16
    3.3.4 Overall Comparison 17
    3.3.5 Analysis 20
    3.3.6 Defending Against Adversarial Attack 21
  3.4 Discussion 22
  3.5 Broader Impact 23
  3.6 Additional Results on Image Classification 23
  3.7 Proof of Theorem 1 24
  3.8 Additional Results of Monte Carlo Simulation 25

Chapter 4 Beyond Preserved Accuracy: Evaluating Loyalty and Robustness of BERT Compression 27
- 4.1 Introduction 27
- 4.2 BERT Compression 29
- 4.3 Metrics Beyond Accuracy 30
  - 4.3.1 Loyalty 30
  - 4.3.2 Robustness 32
- 4.4 Experiments 32
  - 4.4.1 Dataset and Baselines 32
  - 4.4.2 Training Details 33
  - 4.4.3 Experimental Results 33
- 4.5 Combining the Bag of Tricks 34
- 4.6 Conclusion 35
- 4.7 Additional Experimental Results on SST-2 36

Chapter 5 BERT Learns to Teach: Knowledge Distillation with Meta Learning 37
- 5.1 Introduction 37
- 5.2 Knowledge Distillation with Meta Learning 40
  - 5.2.1 Background 40
  - 5.2.2 Methodology 42
- 5.3 Experiments 44
  - 5.3.1 Experimental Setup 44
  - 5.3.2 Experimental Results 47
- 5.4 Analysis 47
  - 5.4.1 Why Does MetaDistil Work? 47
  - 5.4.2 Hyper-parameter Sensitivity 49
  - 5.4.3 Limitation 49
- 5.5 Discussion 50
- 5.6 Additional Results: MetaDistil for Image Classification 50
  - 5.6.1 Experimental Settings 50
  - 5.6.2 Image Recognition Results 51
  - 5.6.3 Static Teaching and Cross Teaching 51
- 5.7 Additional Results: Different Compression Ratios 52
- 5.8 Additional Results: Distillation Dynamics 54
- 5.9 Additional Results: Improvement Analysis 54

Chapter 6 Automatic Multi-Label Prompting: Simple and Interpretable Few-Shot Classification 56
- 6.1 Introduction 56
- 6.2 Prompting for Few-Shot Classification 58
- 6.3 Automatic Multi-Label Prompting 59
  - 6.3.1 Exploiting Multiple Labels 59
  - 6.3.2 Automatic Label Selection 60
- 6.4 Experiments 62

6.4.1 Experimental Setting 62
6.4.2 Experimental Results 65
6.5 Analysis 66
6.5.1 Case Study 66
6.5.2 Ablation Study 67
6.5.3 Scaling Few-Shot Learning 67
6.6 Discussion 68

Chapter 7 InforMask: Unsupervised Informative Masking for Language Model Pre-training 71
7.1 Introduction 71
7.2 Methodology 73
7.2.1 Informative Relevance 74
7.2.2 Scoring Masking Candidates 76
7.2.3 Token-Specific Masking Rates 77
7.3 Experiments 78
7.3.1 Experimental Settings 78
7.3.2 Experimental Results 79
7.3.3 Case Study 85
7.4 Conclusion 86
7.5 Limitations 86

Chapter 8 Baize: An Open-Source Chat Model with Parameter-Efficient Tuning on Self-Chat Data 87
8.1 Introduction 87
8.2 Data Collection via Self-Chat 89
8.3 Model Training 92
8.4 Model Settings 95
8.5 Evaluation 95
8.6 Conclusion and Future Work 97
8.7 Limitations 98

Chapter 9 Related Work 105
9.1 Related Work for BERT Loses Patience 105
9.2 Related Work for MetaDistil 107
9.3 Related Work for Automatic Multi-Label Prompting 109
9.4 Related Work for InforMask 111
9.5 Related Work for Baize 113

Chapter 10 Conclusion and Future Outlook 116
10.1 Future Work for Computational Efficiency 116
10.2 Future Work for Data Efficiency 117

Bibliography 119

LIST OF FIGURES

Figure 3.1. Comparison between Shallow-Deep Net, a prediction score based early exit (threshold is set to 0.9), and our Patience-based Early Exit (patience $t = 2$). 11

Figure 3.2. Analogy between overfitting in training and overthinking in inference. The results are obtained with ALBERT-base on MRPC. 12

Figure 3.3. Speed-accuracy curves of BranchyNet, Shallow-Deep and PABEE on MNLI and SST-2 with ALBERT-base model. 19

Figure 3.4. Accuracy scores and speed-up ratios under different patience settings with ALBERT-base model. The baseline is denoted with gray dash lines. 19

Figure 3.5. Monte Carlo simulation of per PABEE classifier's accuracy vs. the original inference accuracy under different patience settings. 26

Figure 4.1. Three metrics to evaluate the compressed models beyond preserved accuracy. 28

Figure 5.1. The workflow of MetaDistil. 39

Figure 5.2. Results with different student architectures. 48

Figure 5.3. Results with different loss weight α. 48

Figure 5.4. Results with different temperature. 48

Figure 5.5. Learning dynamics of the student and teacher in MetaDistil on the development set of MNLI. 54

Figure 6.1. The illustration of implementing AMuLaP on a binary sentiment classification task (SST-2). 59

Figure 6.2. Comparison of AMuLaP, AMuLaP FT and fine-tuning on MNLI, SST and MRPC with different n for the training set and the development set. 66

Figure 7.1. The informative scores of randomly sampled masking candidates ($s = 4$). . 72

Figure 7.2. The PMI matrix of the words in the sentence 'The dual is between Harry Potter and Lord Voldemort.' 74

Figure 7.3. Macro average MRR of different masking strategies on LAMA, evaluated every 10k steps. 80

Figure 7.4. Performance of InformBERT for the full pretraining process. 81

Figure 7.5. Frequency of common stop words and their corresponding masking rates by InforMask. 82

Figure 7.6. Comparison between looping the same data and using token-specific masking rate to approximate the masking decisions. 83

Figure 7.7. Masking rate of tokens according to different masking policies. 84

Figure 8.1. The pipeline for training Baize and Baize v2. 88

Figure 8.2. An overview of self-distillation with feedback from ChatGPT. 93

Figure 8.3. The performance of Baize models compared with LLaMA Touvron et al. [2023], Alpaca Taori et al. [2023], Vicuna [Chiang et al., 2023] and ChatGPT [OpenAI, 2023a] evaluated by GPT-4 [OpenAI, 2023b]. 94

LIST OF TABLES

Table 3.1. Experimental results (median of 5 runs) of models with ALBERT backbone on the development set and the test set of GLUE. 17

Table 3.2. Experimental results (median of 5 runs) of BERT based models on the development set of GLUE. 18

Table 3.3. Parameter numbers and training time (in minutes) until the best performing checkpoint (on the development set) with and without PABEE on ALBERT and BERT as backbone models. 18

Table 3.4. Experimental results (median of 5 runs) of different sizes of ALBERT on GLUE development set. 21

Table 3.5. Results on the adversarial robustness. 21

Table 3.6. Experimental results (median of 5 runs) of ResNet based models on CIFAR-10 and CIFAR-100 datasets. 23

Table 4.1. Accuracy, loyalty and robustness of compressed models on the test set of MNLI (3 runs). 30

Table 4.2. Accuracy and loyalty for combining multiple compression techniques on the test set of MNLI. 34

Table 4.3. Accuracy and loyalty of some compression techniques on the test set of SST-2. 36

Table 5.1. Experimental results on the development set and the test set of GLUE. 45

Table 5.2. Comparison of training time and memory cost of MetaDistil with the baselines. 50

Table 5.3. Experimental results on the test set of CIFAR-100. 52

Table 5.4. Experimental results of static teaching and cross teaching. 53

Table 5.5. Experimental results on the development set of GLUE in the setting of distilling BERT-base in to BERT_4. 53

Table 5.6. Experimental results on the development set of GLUE in the setting of distilling BERT-large in to BERT_6. 53

Table 6.1. The manual and automatically selected labels by AMuLaP. The templates used for prompting are from Gao et al. [2021b]. 58

Table 6.2. Experimental results under three settings with RoBERTa-large as $\mathscr{L}$. 62

Table 6.3. Most likely label mapping for the SST-2 dataset obtained by PETAL [Schick et al., 2020], Auto-L [Gao et al., 2021b] and our AMuLaP. 64

Table 6.4. Experimental results for the ablation study. 65

Table 7.1. Statistics of LAMA [Petroni et al., 2019]. 76

Table 7.2. Statistics of SQuAD v1 and v2 [Rajpurkar et al., 2016, 2018]. 76

Table 7.3. Performance of different masking strategies and models on LAMA [Petroni et al., 2019]. 77

Table 7.4. Performance on SQuAD v1 and v2 [Rajpurkar et al., 2016, 2018] development set. 78

Table 7.5. Some examples of InformBERT and RoBERTa-base predictions on LAMA. 84

Table 8.1. *(Not cherry-picked)* An example of self-chat generated by ChatGPT [OpenAI, 2023a] with a seed sampled from the Quora dataset. 90

Table 8.2. Statistics of the number of dialogues, average number of turns, and response lengths of each turn. 91

Table 8.3. Data, numbers of parameters and training time for training Baize models. . 91

Table 8.4. Performance on LM Evaluation Harness [Gao et al., 2021a], evaluated by Hugging Face. 96

Table 8.5. *(Not cherry-picked)* An example of asking chat models to analyze the Lehman Brothers' bankruptcy. 100

Table 8.6. *(Not cherry-picked)* An example of asking chat models to explain a joke. . . 101

Table 8.7. *(Not cherry-picked)* Examples of how chat models respond to unethical requests from users. 102

Table 8.8. *(Cherry-picked)* An example of a coding question. 103

Table 8.9. *(Not cherry-picked)* An example of Baize-Healthcare answering a healthcare question. 104

ACKNOWLEDGEMENTS

I am profoundly grateful to the many individuals who have supported and guided me throughout this remarkable journey.

Foremost, I extend my deepest appreciation to my advisor, Julian McAuley, for his unwavering support and guidance over the years. His mentorship has not only honed my research skills but also equipped me with the resilience to navigate the challenges, stress, and uncertainties inherent in academic research. His empathy, insightfulness, and unwavering belief in my potential have been instrumental in shaping my career path.

My sincere thanks also go to my thesis committee members, Taylor Berg-Kirkpatrick, Zhiting Hu, and Jingbo Shang, for their valuable suggestions and encouragement.

The financial support from the NSF Award #1750063 and the Jacobs School of Engineering Powell Fellowship has been crucial to my research endeavors, and for this, I am sincerely thankful.

Collaborating with Wangchunshu Zhou, Daya Guo, Han Wang, Penghan Wang, Jiaxin Pei, Tianyang Liu, Ryan Tran, Nafis Sadeq, Zexue He, Zhankui He, and many others has been an honor and a privilege. Your partnership has significantly enriched my research experience.

A heartfelt thank you to all members of The McAuley Lab at UCSD. Your camaraderie and support have always made the lab feel like a second home.

I am grateful for the invaluable experiences and learnings gained during my internships and part-time roles at Microsoft Research Asia, Hugging Face, Google, Azure Cognitive Services, Microsoft Research Redmond, and Boson AI. Working alongside such talented teams has undoubtedly enhanced my research capabilities.

Special appreciation goes to Professors Chenliang Li and Zhenzhong Chen at Wuhan University, who ignited my passion for research and laid the foundation for my academic pursuits.

To all my friends at UCSD, your presence has added vibrant colors to my life. I am particularly grateful to Shuheng Li and Jiashun Wang for their enduring support and companionship.

My heartfelt thanks to friends from my past who stay around through the years. Thanks

to my friends-turned-family, Guande and Chen. Thanks to my undergraduate roommates, Zixiao, Boxue, and Tianzong, who have always been a delight in my life. Thanks to all my friends from undergraduate and high school. While I am unable to list everyone, please know that each of you has played a cherished role in my life, and for that, I am deeply grateful.

I am also indebted to my peers from the NLP and ML community. Our interactions, both online and at conferences, have been a source of inspiration and a highlight of my PhD journey.

Lastly, the love and support from my parents and grandparents have been the cornerstone of my achievements. During the challenging period of the COVID-19 lockdown, and continuing beyond, my two pet cats, Shirley and Simba, have been a source of immense comfort and joy. Thanks to all of you!

The abstract, Chapter 1, 2, and 10, in part, are a reprint of the material as it appears in "A Survey on Model Compression and Acceleration for Pretrained Language Models" by Canwen Xu and Julian McAuley, which was published at *AAAI Conference on Artificial Intelligence*, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 3 and 9, in part, is a reprint of the material as it appears in "BERT Loses Patience: Fast and Robust Inference with Early Exit" by Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei, which was published at *Conference on Neural Information Processing Systems*, 2020. The dissertation author was the co-primary investigator and co-first author of this paper.

Chapter 4, in part, is a reprint of the material as it appears in "Beyond Preserved Accuracy: Evaluating Loyalty and Robustness of BERT Compression" by Canwen Xu, Wangchunshu Zhou, Tao Ge, Ke Xu, Julian McAuley, and Furu Wei, which was published at *Empirical Methods in Natural Language Processing*, 2021. The dissertation author was the co-primary investigator and co-first author of this paper.

Chapter 5 and 9, in part, is a reprint of the material as it appears in "BERT Learns to Teach: Knowledge Distillation with Meta Learning" by Wangchunshu Zhou, Canwen Xu, and Julian McAuley, which was published at *Association for Computational Linguistics*, 2022. The dissertation author was the co-primary investigator and co-first author of this paper.

Chapter 6 and 9, in part, is a reprint of the material as it appears in "Automatic Multi-Label Prompting: Simple and Interpretable Few-Shot Classification" by Han Wang, Canwen Xu, and Julian McAuley, which was published at *North American Chapter of the Association for Computational Linguistics*, 2022. The dissertation author was the co-primary investigator and co-first author of this paper.

Chapter 7 and 9, in part, is a reprint of the material as it appears in "InforMask: Unsupervised Informative Masking for Language Model Pretraining" by Nafis Sadeq, Canwen Xu, and Julian McAuley, which was published at *Empirical Methods in Natural Language Processing*, 2022. The dissertation author was the co-primary investigator and co-first author of this paper.

Chapter 8 and 9, in part, is a reprint of the material as it appears in "Baize: An Open-Source Chat Model with Parameter-Efficient Tuning on Self-Chat Data" by Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley, which was published at *Empirical Methods in Natural Language Processing*, 2023. The dissertation author was the co-primary investigator and co-first author of this paper.

VITA

2020	B.Eng. in Computer Science & Technology, Wuhan University
2022	M.S. in Computer Science, University of California San Diego
2024	Ph.D. in Computer Science, University of California San Diego

PUBLICATIONS

See the full list of publications on my website: https://www.canwenxu.net/publications.

ABSTRACT OF THE DISSERTATION

Efficient Natural Language Processing for Language Models

by

Canwen Xu

Doctor of Philosophy in Computer Science

University of California San Diego, 2024

Professor Julian McAuley, Chair

Despite achieving state-of-the-art performance on many NLP tasks, the high energy cost and long inference delay prevent Transformer-based language models (LMs) from seeing broader adoption including for edge and mobile computing. Our efficient NLP research aims to comprehensively consider computation, time and carbon emission for the entire life-cycle of NLP, including data preparation, model training and inference.

We demonstrate ways to promote computational efficiency in natural language processing, thus reducing hardware and software bottlenecks of training and inference, which is crucial in applying such models in production. Efficient NLP further facilitates democratization of language technology and allows language models to be accessible to more people.

Chapter 1

Introduction

The recent success of applying pretrained deep Transformers [Vaswani et al., 2017] on different NLP tasks [Devlin et al., 2019, Raffel et al., 2020, Le Scao et al., 2022] has raised concerns about its efficiency. The high computational cost also prevents these language models (LMs) from being deployed in production [Sun et al., 2020b]. To address this problem, efficient natural language processing (Efficient NLP) refers to techniques that aim to make training and inference of an ML model faster (time-efficient), consume fewer computational resources (computation-efficient), less memory (memory-efficient), less disk space (storage-efficient) and use less human-labeled data (data-efficient). For inference, one popular class of techniques is *model compression and acceleration*, where a large and slow model is compressed to a lightweight model that can be stored with limited disk space on a mobile device, or accelerated to run with low latency (or both). Also, training a large model and then compressing it to a small one can be efficient for training and good for generalization [Li et al., 2020b]. For training, efficient methods aim to reduce the training time and GPU memory usage to allow more throughput. Another line of research aims to use less or auto-generated data to train language models.

In addition to technical considerations, large models also raise environmental and ethical concerns [Bender et al., 2021]. Large models have a high carbon footprint which an efficient model can reduce, potentially with little sacrifice in performance. Meanwhile, large models set obstacles for engineers and researchers from developing countries who cannot afford the necessary hardware for running the model [Bender et al., 2021]. Thus, efficient NLP techniques can be critical to make state-of-the-art NLP techniques more accessible and facilitate inclusiveness. My research stands at the heart of the following question:

How can we make expensive NLP technology more accessible to everyone?

1.1 Computation Efficiency

In the quest for efficiency in natural language processing, a critical area of focus is efficient inference. The goal here is to optimize the deployment of language models so they can operate swiftly and with minimal resource consumption, especially in environments with constrained computational power, such as mobile devices or embedded systems.

A primary technique in achieving efficient inference is through model compression and acceleration. This involves transforming large, cumbersome models into more compact versions without significantly sacrificing performance. Compressed models demand less storage and memory, making them suitable for devices with limited capacity. Acceleration techniques further ensure that these models can execute tasks rapidly, offering low-latency responses essential for real-time applications.

In addition to traditional model compression methods (e.g., pruning, knowledge distillation), an innovative approach to expedite inference is the implementation of early exit strategies. These techniques allow a model to provide answers at intermediate stages of processing, rather than completing the entire set of computations. This can significantly reduce inference time, especially for simpler queries that do not require deep processing layers. Early exit strategies strike a balance between speed and accuracy, offering a robust solution in scenarios where response time is crucial.

1.2 Data Efficiency

Another vital area of research in efficient NLP is data efficiency. The traditional approach to training language models requires vast amounts of data, which is not always feasible or available. Data-efficient NLP focuses on strategies to reduce this dependency, enabling the training of powerful models with limited data or even datasets generated by other models.

One avenue is the exploration of unsupervised learning techniques with different weights for different data. These methods involve intelligently selecting parts of the input data to be masked during training, encouraging the model to learn more from less data. Such strategies can enhance the model's ability to understand and generalize from the available information, reducing the need for extensive pretraining datasets.

Another approach is the use of few-shot prompt learning. This method allows a model to handle classification tasks with minimal training data. By automatically mapping labels to a few tokens, the model can be guided to provide accurate classifications in few-shot learning scenarios.

Additionally, a facet of data-efficient training is the incorporation of data generation techniques. This method involves generating new data through the model's own dialogues, thereby creating a rich and diverse dataset for supervised fine-tuning. By employing self-generated data, the model can adapt more effectively to various contexts and user inputs, handling a wide range of queries with minimal computational effort.

1.3 Summary

While Chapter 1 introduces introduces the challenges and necessity of efficient NLP, Chapter 2 lays foundation about the basics like what is a language model and how to measure its efficiency.

Chapter 3, 4, and 5 focus on efficient inference of language models.

Chapter 3 introduces a dynamic acceleration approach for pretrained language models.

This method terminates inference earlier when a patience-based criterion is met.

Chapter 4 discusses different types of methods for compressing a pretrained language model and presents a new metric to measure the success of model compression.

Chapter 5 presents a new knowledge distillation approach for language models that uses meta learning to improve the teacher model while teaching a student model.

Chapter 6, 7, and 8 concern data-efficient natural language processing.

Chapter 6 introduces a simple yet effective way of leveraging few-shot learning ability of language models with prompt learning.

Chapter 7 presents a new method for deciding masks for masked language models. This method assigns importance to tokens based on their semantic associations, thus help the model learn better with less data and fewer training steps.

Chapter 8 introduces a pioneering work on fine-tuning large language model with pure synthetic data by allowing a teacher model to chat to itself.

Chapter 9 refers to the related work for all novel work introduced in previous chapter.

Chapter 10 concludes the dissertation while discussing the novel possibilities and future outlook towards efficient NLP for language models.

This chapter, in part, are a reprint of the material as it appears in “A Survey on Model Compression and Acceleration for Pretrained Language Models” by Canwen Xu and Julian McAuley, which was published at *AAAI Conference on Artificial Intelligence*, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 2

Background

Language models have revolutionized the way we interact with technology and process natural language. The advances in model architecture, particularly with Transformer-based models, have led to significant improvements in understanding and generating human language. These models are not only pivotal in academic research but have also found widespread application in various industries, from automated translation services to voice-activated assistants. However, as these models grow in complexity and size, it becomes increasingly important to consider factors like computational efficiency, environmental impact, and accessibility. The challenge moving forward is to balance the quest for more powerful models with the practical constraints of deploying these models in real-world scenarios. Future developments must focus on optimizing performance while minimizing resource usage and environmental impact, ensuring that the benefits of language models are accessible to a broader range of users and applications.

2.1 Language Models

Language models aim to model the language, typically by predicting the next token based on the given prompt. A language model learns to assign the probability of a sequence of tokens $X = x_1, \ldots, x_T$ using conditional probability:

$$P(X) = \prod_{t=1}^{T} P(x_t | x_{1:t-1}). \tag{2.1}$$

Such models are trained with a cross-entropy loss between ground truth tokens and predicted probability distribution given current tokens for the next time step.

Alternatively, some language models (e.g., BERT [Devlin et al., 2019]) predict a masked token in a sequence, instead of the next token. A popular architecture of language models is Transformer and its variants, renowned for their effectiveness in processing sequential data. At its core, the Transformer model eschews traditional recurrent layers, instead relying on a self-attention mechanism that computes the relevance of each part of the input data to other parts. This architecture consists of an encoder and a decoder, each comprising multiple identical layers. Within each layer, the key component is the multi-head attention mechanism, which enables the model to focus on different parts of the input sequence simultaneously, a feature that is crucial for understanding the context and relationships in the data. This mechanism is often represented as $\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$, where Q,K,V are queries, keys, and values, respectively, and d_k is the dimension of the keys. Additionally, each layer includes a position-wise feed-forward network and employs residual connections and layer normalization to facilitate training of deep networks. Positional encoding is added to the input embeddings to provide the model with information about the order of the sequence, compensating for the lack of recurrence or convolution. This architecture allows the Transformer to efficiently handle dependencies and patterns in sequential data, making it a powerful tool for language modeling.

2.2 Measuring Efficiency of Language Models

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

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

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

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

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

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

Memory Footprint refers to the amount of memory required to store and execute a model. This includes both the storage space needed for the model parameters and the runtime memory needed for processing inputs. A smaller memory footprint is particularly crucial for applications with limited memory resources, like mobile and embedded systems. Memory footprint is an important

---

[1] https://codecarbon.io/

consideration for large models, as it can limit the model's applicability in resource-constrained environments. Moreover, it can also affect the scalability of a model in distributed systems or in scenarios where multiple models need to run concurrently.

2.3 Conclusion

In this chapter, we have explored the intricacies of language models, delving into their structure, operational mechanisms, and the various measures used to assess their efficiency. The evolution from simple predictive models to sophisticated Transformer-based architectures underscores the rapid advancements in the field of natural language processing. While these developments have opened up new horizons in understanding and interacting with human language, they also bring to the forefront critical considerations regarding computational resources, environmental sustainability, and practical applicability. As we continue to push the boundaries of what is possible with language models, it is imperative to address these challenges, ensuring that the progress in this field is sustainable and beneficial for a wide array of applications. The future of language models lies in striking a balance between sophistication and efficiency, harnessing their potential while being mindful of their impact on resources and the environment.

This chapter, in part, are a reprint of the material as it appears in "A Survey on Model Compression and Acceleration for Pretrained Language Models" by Canwen Xu and Julian McAuley, which was published at *AAAI Conference on Artificial Intelligence*, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 3

BERT Loses Patience: Fast and Robust Inference with Early Exit

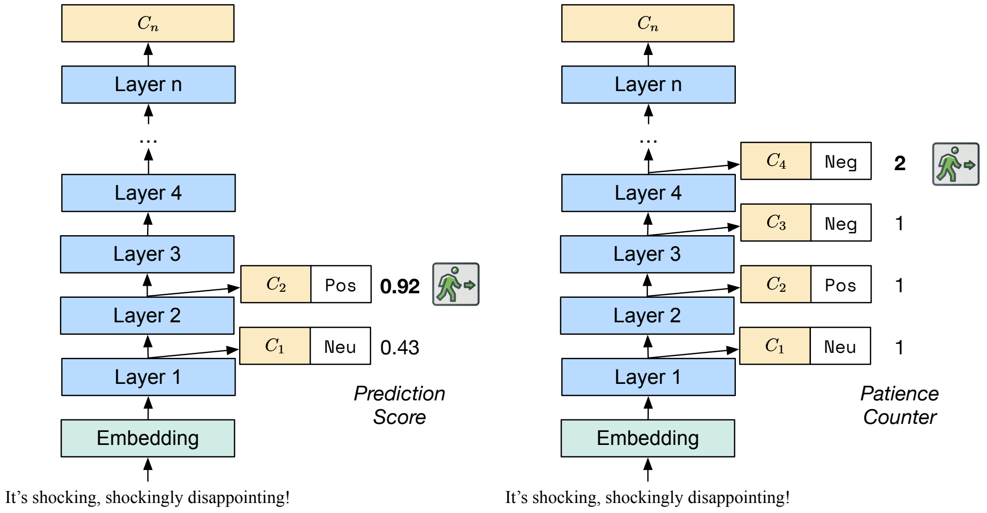
In this chapter, we propose Patience-based Early Exit, a straightforward yet effective inference method that can be used as a plug-and-play technique to simultaneously improve the efficiency and robustness of a pretrained language model (PLM). To achieve this, our approach couples an internal-classifier with each layer of a PLM and dynamically stops inference when the intermediate predictions of the internal classifiers do not change for a pre-defined number of steps. Our approach improves inference efficiency as it allows the model to make a prediction with fewer layers. Meanwhile, experimental results with an ALBERT model show that our method can improve the accuracy and robustness of the model by preventing it from overthinking and exploiting multiple classifiers for prediction, yielding a better accuracy-speed trade-off compared to existing early exit methods.

3.1 Introduction

In Natural Language Processing (NLP), pretraining and fine-tuning have become a new norm for many tasks. Pretrained language models (PLMs) (e.g., BERT [Devlin et al., 2019], XLNet [Yang et al., 2019], RoBERTa [Liu et al., 2019b], ALBERT [Lan et al., 2020]) contain many layers and millions or even billions of parameters, making them computationally expensive and inefficient regarding both memory consumption and latency. This drawback hinders their application in scenarios where inference speed and computational costs are crucial. Another bottleneck of overparameterized PLMs that stack dozens of Transformer layers is the "overthinking" problem [Kaya et al., 2019] during their decision-making process. That is, for many input samples, their shallow representations at an earlier layer are adequate to make a correct classification, whereas the representations in the final layer may be otherwise distracted by over-complicated or irrelevant features that do not generalize well. The overthinking problem in PLMs leads to wasted computation, hinders model generalization, and may also make them vulnerable to adversarial attacks [Jin et al., 2020].

In this chapter, we propose a novel **Pa**tience-**b**ased **E**arly **E**xit (PABEE) mechanism to enable models to stop inference dynamically. PABEE is inspired by the widely used Early Stopping [Morgan and Bourlard, 1989, Prechelt, 1998] strategy for model training. It enables better input-adaptive inference of PLMs to address the aforementioned limitations. Specifically, our approach couples an internal classifier with each layer of a PLM and dynamically stops inference when the intermediate predictions of the internal classifiers do not change t times consecutively (see Figure 3.1b), where t is a pre-defined patience. We first show that our method is able to improve the accuracy compared to conventional inference under certain assumptions. Then we conduct extensive experiments on the GLUE benchmark and show that PABEE outperforms existing prediction probability distribution-based exit criteria by a large margin. In addition, PABEE can simultaneously improve inference speed and adversarial robustness of the original model while retaining or even improving its original accuracy with minor additional effort in terms of model size and training time. Also, our method can dynamically adjust the accuracy-efficiency trade-off to fit different devices and resource constraints by tuning the patience hyperparameter without retraining the model, which is favored in real-world applications [Cai et al., 2020]. Although we focus on PLM in this chapter, we also have conducted experiments on image classification tasks with the popular ResNet [He et al., 2016] as the backbone model and present the results in Section 3.6 to verify the generalization ability of PABEE.

To summarize, our contribution is two-fold: (1) We propose Patience-based Early Exit,



(a) Shallow-Deep Net [Kaya et al., 2019] **(b)** Patience-based Early Exit (PABEE)

Figure 3.1. Comparison between Shallow-Deep Net, a prediction score based early exit (threshold is set to 0.9), and our Patience-based Early Exit (patience $t = 2$). A classifier is denoted by C_i, and n is the number of layers in a model. In this figure, Shallow-Deep incorrectly exits based on the prediction score while PABEE considers multiple classifiers and exits with a correct prediction.

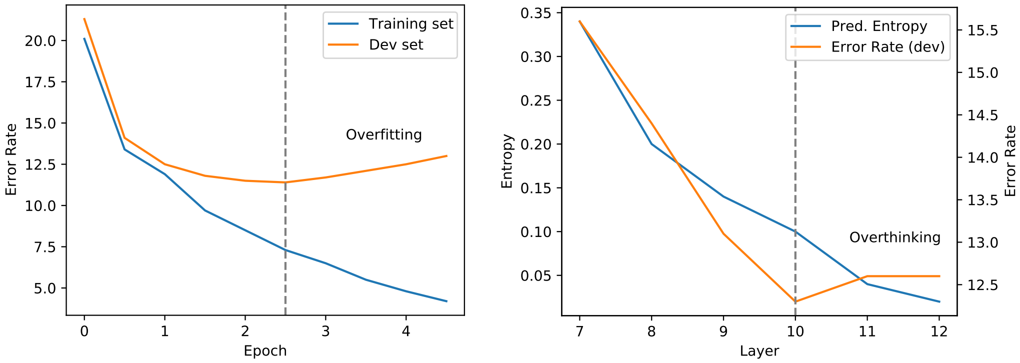
a novel and effective inference mechanism and show its feasibility of improving the efficiency and the accuracy of deep neural networks with theoretical analysis. (2) Our empirical results on the GLUE benchmark highlight that our approach can simultaneously improve the accuracy and robustness of a competitive ALBERT model, while speeding up inference across different tasks with trivial additional training resources in terms of both time and parameters.

3.2 Patience-based Early Exit

Patience-based Early Exit (PABEE) is a plug-and-play method that can work well with minimal adjustment on training.

3.2.1 Motivation

We first conduct experiments to investigate the overthinking problem in PLMs. As shown in Figure 3.2b, we illustrate the prediction distribution entropy [Teerapittayanon et al., 2016] and the error rate of the model on the development set as more layers join the prediction.



(a) Overfitting in training **(b)** Overthinking in inference

Figure 3.2. Analogy between overfitting in training and overthinking in inference. The results are obtained with ALBERT-base on MRPC.

Although the model becomes more "confident" (lower entropy indicates higher confidence in BranchyNet [Teerapittayanon et al., 2016]) with its prediction as more layers join, the actual error rate instead increases after 10 layers. This phenomenon was discovered and named "overthinking" by Kaya et al. [2019]. Similarly, as shown in Figure 3.2a, after 2.5 epochs of training, the model continues to get better accuracy on the training set but begins to deteriorate on the development set. This is the well-known overfitting problem which can be resolved by applying an early stopping mechanism [Morgan and Bourlard, 1989, Prechelt, 1998]. From this aspect, overfitting in training and overthinking in inference are naturally alike, inspiring us to adopt an approach similar to early stopping for inference.

3.2.2 Inference

The inference process of PABEE is illustrated in Figure 3.1b. Formally, we define a common inference process as the input instance $\mathbf{x}$ goes through layers $L_1 \dots L_n$ and the classifier/regressor C_n to predict a class label distribution $\mathbf{y}$ (for classification) or a value y (for regression, we assume the output dimension is 1 for brevity). We couple an internal classifier/regressor $C_1 \dots C_{n-1}$ with each layer of $L_1 \dots L_{n-1}$, respectively. For each layer L_i, we first calculate its hidden state $\mathbf{h}_i$:

$$\begin{aligned} \mathbf{h}_i &= L_i(\mathbf{h}_{i-1}) \\ \mathbf{h}_0 &= \text{Embedding}(\mathbf{x}) \end{aligned} \tag{3.1}$$

Then, we use its internal classifier/regressor to output a distribution or value as a per-layer prediction $\mathbf{y}_i = C_i(\mathbf{h}_i)$ or $y_i = C_i(\mathbf{h}_i)$. We use a counter *cnt* to store the number of times that the predictions remain "unchanged". For classification, cnt_i is calculated by:

$$cnt_i = \begin{cases} cnt_{i-1} + 1 & \arg\max(\mathbf{y}_i) = \arg\max(\mathbf{y}_{i-1}), \\ 0 & \arg\max(\mathbf{y}_i) \neq \arg\max(\mathbf{y}_{i-1}) \vee i = 0. \end{cases} \tag{3.2}$$

While for regression, cnt_i is calculated by:

$$cnt_i = \begin{cases} cnt_{i-1} + 1 & y_i - y_{i-1} < \tau, \\ 0 & y_i - y_{i-1} \geq \tau \vee i = 0. \end{cases} \tag{3.3}$$

where τ is a pre-defined threshold. We stop inference early at layer L_j when $cnt_j = t$. If this condition is never fulfilled, we use the final classifier C_n for prediction. In this way, the model can exit early without passing through all layers to make a prediction.

As shown in Figure 3.1a, prediction score-based early exit relies on the softmax score. As revealed by prior work [Szegedy et al., 2014, Jiang et al., 2018], prediction of probability distributions (i.e., softmax scores) suffers from being over-confident to one class, making it an unreliable metric to represent confidence. Nevertheless, the capability of a low layer may not match its high confidence score. In Figure 3.1a, the second classifier outputs a high confidence score and incorrectly terminates inference. With Patience-based Early Exit, the stopping criteria is in a cross-layer fashion, preventing errors from one single classifier. Also, since PABEE comprehensively considers results from multiple classifiers, it can also benefit from an Ensemble

Learning [Krogh and Vedelsby, 1994] effect.

3.2.3 Training

PABEE requires that we train internal classifiers to predict based on their corresponding layers' hidden states. For classification, the loss function $\mathscr{L}_i$ for classifier C_i is calculated with Cross Entropy:

$$\mathscr{L}_i = -\sum_{z \in Z} \left[\mathbb{1}\left[\mathbf{y}_i = z\right] \cdot \log P\left(\mathbf{y}_i = z | \mathbf{h}_i\right) \right] \tag{3.4}$$

where z and Z denote a class label and the set of class labels, respectively. For regression, the loss is instead calculated by a (mean) squared error:

$$\mathscr{L}_i = \left(y_i - \hat{y}_i\right)^2 \tag{3.5}$$

where $\hat{y}$ is the ground truth. Then, we calculate and train the model to minimize the total loss $\mathscr{L}$ by a weighted average following Kaya et al. [2019]:

$$\mathscr{L} = \frac{\sum_{j=1}^{n} j \cdot \mathscr{L}_j}{\sum_{j=1}^{n} j} \tag{3.6}$$

In this way, every possible inference branch has been covered in the training process. Also, the weighted average can correspond to the relative inference cost of each internal classifier.

3.2.4 Theoretical Analysis

It is straightforward to see that Patience-based Early Exit is able to reduce inference latency. To understand whether and under what conditions it can also improve accuracy, we conduct a theoretical comparison of a model's accuracy with and without PABEE. We consider the case of binary classification for simplicity and conclude that:

Theorem 1 *Assuming the patience of PABEE inference is t, the total number of internal classifiers (IC) is n, the misclassification probability (i.e., error rate) of all internal classifiers* *(excluding the final classifier) is q, and the misclassification probability of the final classifier and the original classifier (without ICs) is p. Then the PABEE mechanism improves the accuracy of conventional inference as long as* $n - t < (\frac{1}{2q})^t(\frac{p}{q}) - p$

(the proof is detailed in Section 3.7).

We can see the above inequality can be easily satisfied in practice. For instance, when $n = 12$, $q = 0.2$, and $p = 0.1$, the above equation is satisfied as long as the patience $t \geq 4$. Additionally, we verify the statistical feasibility of PABEE with Monte Carlo simulation in Section 3.8. To further test PABEE with real data and tasks, we also conduct extensive experiments in the following section.

3.3 Experiments

3.3.1 Tasks and Datasets

We evaluate our proposed approach on the GLUE benchmark [Wang et al., 2019b]. Specifically, we test on Microsoft Research Paraphrase Matching (MRPC) [Dolan and Brockett, 2005], Quora Question Pairs (QQP)[1] and STS-B [Conneau and Kiela, 2018] for Paraphrase Similarity Matching; Stanford Sentiment Treebank (SST-2) [Socher et al., 2013] for Sentiment Classification; Multi-Genre Natural Language Inference Matched (MNLI-m), Multi-Genre Natural Language Inference Mismatched (MNLI-mm) [Williams et al., 2018], Question Natural Language Inference (QNLI) [Rajpurkar et al., 2016] and Recognizing Textual Entailment (RTE) [Wang et al., 2019b] for the Natural Language Inference (NLI) task; The Corpus of Linguistic Acceptability (CoLA) [Warstadt et al., 2019] for Linguistic Acceptability. We exclude WNLI [Levesque, 2011] from GLUE following previous work [Devlin et al., 2019, Jiao et al., 2020, Xu et al., 2020].

---

[1] https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs

3.3.2 Baselines

For GLUE tasks, we compare our approach with four types of baselines: **(1) Backbone models:** We choose ALBERT-base and BERT-base, which have approximately the same inference latency and accuracy. **(2) Directly reducing layers:** We experiment with the first 6 and 9 layers of the original (AL)BERT with a single output layer on the top, denoted by (AL)BERT-6L and (AL)BERT-9L, respectively. These two baselines help to set a lower bound for methods that do not employ any technique. **(3) Static model compression approaches:** For pruning, we include the results of LayerDrop [Fan et al., 2020] and attention head pruning [Michel et al., 2019] on ALBERT. For reference, we also report the performance of state-of-the-art methods on compressing the BERT-base model with knowledge distillation or module replacing, including DistillBERT [Sanh et al., 2019], BERT-PKD [Sun et al., 2019a] and BERT-of-Theseus [Xu et al., 2020]. **(4) Input-adaptive inference:** Following the settings in concurrent studies [Schwartz et al., 2020b, Liu et al., 2020b, Xin et al., 2020b], we add internal classifiers after each layer and apply different early exit criteria, including that employed by BranchyNet [Teerapittayanon et al., 2016] and Shallow-Deep [Kaya et al., 2019]. We also add DeeBERT [Xin et al., 2020b], a BranchyNet variant on BERT alongside our BranchyNet implementation. To make a fair comparison, the internal classifiers and their insertions are exactly same in both baselines and Patience-based Early Exit. We search over a set of thresholds to find the one delivering the best accuracy for the baselines while targeting a speed-up ratio between 1.30× and 1.96× (the speed-up ratios of (AL)BERT-9L and -6L, respectively).

3.3.3 Experimental Setting

Training We add a linear layer after each intermediate layer of the pretrained BERT/ALBERT model as the internal classifiers. We perform grid search over batch sizes of {16, 32, 128}, and learning rates of {1e-5, 2e-5, 3e-5, 5e-5} with an Adam optimizer. We apply an early stopping mechanism and select the model with the best performance on the development set. We conduct

Table 3.1. Experimental results (median of 5 runs) of models with ALBERT backbone on the development set and the test set of GLUE. The numbers under each dataset indicate the number of training samples. The acceleration ratio is averaged across 8 tasks. We mark "-" on STS-B for BranchyNet and Shallow-Deep since they do not support regression.

Method	**#Param**	**Speed -up**	**CoLA** (8.5K)	**MNLI** (393K)	**MRPC** (3.7K)	**QNLI** (105K)	**QQP** (364K)	**RTE** (2.5K)	**SST-2** (67K)	**STS-B** (5.7K)	**Macro Score**
Dev. Set											
ALBERT-base	12M	1.00×	58.9	84.6	89.5	91.7	89.6	78.6	92.8	89.5	84.4
ALBERT-6L	12M	1.96×	53.4	80.2	85.8	87.2	86.8	73.6	89.8	83.4	80.0
ALBERT-9L	12M	1.30×	55.2	81.2	87.1	88.7	88.3	75.9	91.3	87.1	81.9
LayerDrop	12M	1.96×	53.6	79.8	85.9	87.0	87.3	74.3	90.7	86.5	80.6
HeadPrune	12M	1.22×	54.1	80.3	86.2	86.8	88.0	75.1	90.5	87.4	81.1
BranchyNet	12M	1.88×	55.2	81.7	87.2	88.9	87.4	75.4	91.6	-	-
Shallow-Deep	12M	1.95×	55.5	81.5	87.1	89.2	87.8	75.2	91.7	-	-
PABEE *(ours)*	12M	1.57×	**61.2**	**85.1**	**90.0**	**91.8**	**89.6**	**80.1**	**93.0**	**90.1**	**85.1**
Test Set											
ALBERT-base	12M	1.00×	54.1	84.3	87.0	90.8	71.1	76.4	94.1	85.5	80.4
PABEE *(ours)*	12M	1.57×	**55.7**	**84.8**	**87.4**	**91.0**	**71.2**	**77.3**	**94.1**	**85.7**	**80.9**

our experiments on a single Nvidia V100 16GB GPU.

Inference Following prior work on input-adaptive inference [Teerapittayanon et al., 2016, Kaya et al., 2019], inference is on a per-instance basis, i.e., the batch size for inference is set to 1. This is a common latency-sensitive production scenario when processing individual requests from different users [Schwartz et al., 2020b]. We report the median performance over 5 runs with different random seeds because the performance on relatively small datasets such as CoLA and RTE usually has large variance. For PABEE, we set the patience $t = 6$ in the overall comparison to keep the speed-up ratio between 1.30× and 1.96× while obtaining good performance following Figure 3.4. We further analyze the behavior of the PABEE mechanism with different patience settings in Section 3.3.5.

3.3.4 Overall Comparison

We first report our main result on GLUE with ALBERT as the backbone model in Table 3.1. This choice is made because: (1) ALBERT is a state-of-the-art PLM for natural language understanding. (2) ALBERT is already very efficient in terms of the number of parameters

Table 3.2. Experimental results (median of 5 runs) of BERT based models on the development set of GLUE. We mark "-" on STS-B for BranchyNet and Shallow-Deep since they do not support regression.

Method	**#Param**	**Speed -up**	**MNLI** (393K)	**SST-2** (67K)	**STS-B** (5.7K)
BERT-base	108M	1.00×	84.5	92.1	88.9
BERT-6L	66M	1.96×	80.1	89.6	81.2
BERT-9L	87M	1.30×	81.4	90.5	85.0
DistilBERT	66M	1.96×	79.0	90.7	81.2
BERT-PKD	66M	1.96×	81.3	91.3	86.2
BERT-of-Theseus	66M	1.96×	82.3	91.5	**88.7**
BranchyNet	108M	1.87×	80.3	90.4	-
DeeBERT	108M	1.59×	80.7	90.0	-
Shallow-Deep	108M	1.91×	80.5	90.6	-
PABEE *(ours)*	108M	1.62×	**83.6**	**92.0**	**88.7**

Table 3.3. Parameter numbers and training time (in minutes) until the best performing checkpoint (on the development set) with and without PABEE on ALBERT and BERT as backbone models.

Method	#Param MNLI	#Param SST-2	Train. time (min) MNLI	Train. time (min) SST-2
ALBERT				
w/o PABEE	12M	12M	234	113
w/ PABEE	+36K	+24K	**227**	**108**
BERT				
w/o PABEE	108M	108M	247	121
w/ PABEE	+36K	+24K	**242**	**120**

and memory use because of its layer sharing mechanism, but still suffers from the problem of high inference latency. We can see that our approach significantly outperforms all compared approaches on improving the inference efficiency of PLMs, demonstrating the effectiveness of the proposed PABEE mechanism. Surprisingly, our approach consistently improves the performance of the original ALBERT model by a relatively large margin while speeding-up inference by 1.57×. This is, to the best of our knowledge, the first inference strategy that can improve both the speed and performance of a fine-tuned PLM.

To better compare the efficiency of PABEE with the method employed in BranchyNet and Shallow-Deep, we illustrate speed-accuracy curves in Figure 3.3 with different trade-off hyperparameters (i.e., threshold for BranchyNet and Shallow-Deep, patience for PABEE). Notably, PABEE retains higher accuracy than BranchyNet and Shallow-Deep under the same speed-up ratio, showing its superiority over prediction score based methods.

To demonstrate the versatility of our method with different PLMs, we report the results on a representative subset of GLUE with BERT [Devlin et al., 2019] as the backbone model in Table 3.2. We can see that our BERT-based model significantly outperforms other compared methods with either knowledge distillation or prediction probability based input-adaptive inference

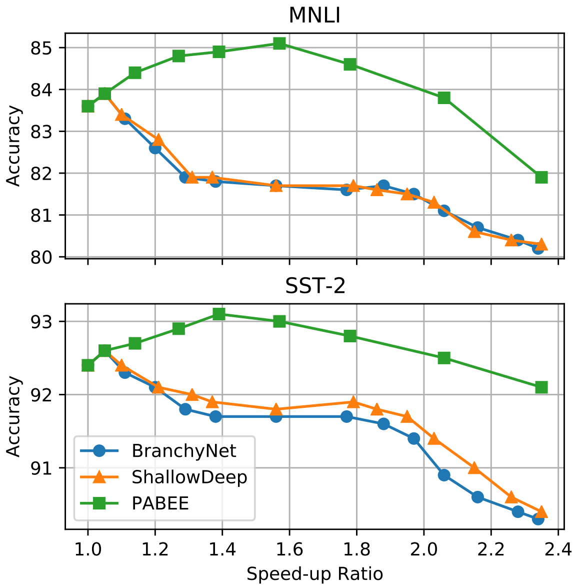


Figure 3.3. Speed-accuracy curves of BranchyNet, Shallow-Deep and PABEE on MNLI and SST-2 with ALBERT-base model.

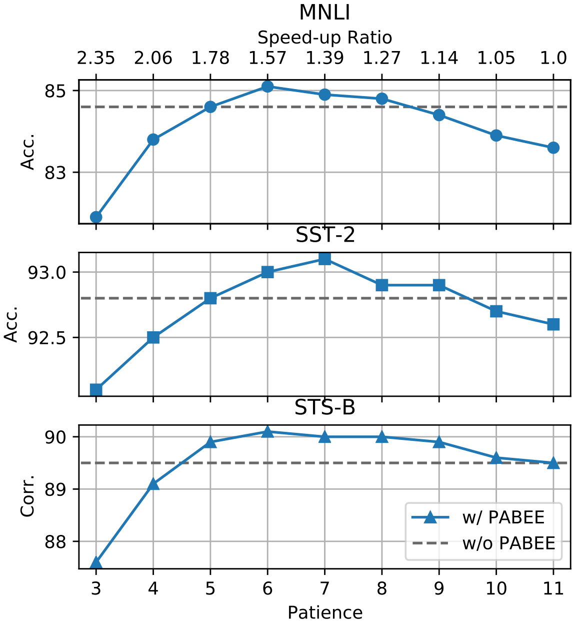


Figure 3.4. Accuracy scores and speed-up ratios under different patience settings with ALBERT-base model. The baseline is denoted with gray dash lines.

methods. Notably, the performance is slightly lower than the original BERT model while PABEE improves the accuracy on ALBERT. We suspect that this is because the intermediate layers of BERT have never been connected to an output layer during pretraining, which leads to a mismatch between pretraining and fine-tuning when adding the internal classifiers. However, PABEE still has a higher accuracy than various knowledge distillation-based approaches as well as prediction probability distribution based models, showing its potential as a generic method for deep neural networks of different kinds.

As for the cost of training, we present parameter numbers and training time with and without PABEE with both BERT and ALBERT backbones in Table 3.3. Although more classifiers need to be trained, training PABEE is no slower (even slightly faster) than conventional fine-tuning, which may be attributed to the additional loss functions of added internal classifiers. This makes our approach appealing compared with other approaches for accelerating inference such as pruning or distillation because they require separately training another model for each speed-up ratio in addition to training the full model. Also, PABEE only introduces fewer than 40K parameters (0.33% of the original 12M parameters).

3.3.5 Analysis

Impact of Patience

As illustrated in Figure 3.4, different patience can lead to different speed-up ratios and performance. For a 12-layer ALBERT model, PABEE reaches peak performance with a patience of 6 or 7. On MNLI, SST-2 and STS-B, PABEE can always outperform the baseline with patience between 5 and 8. Notably, unlike BranchyNet and Shallow-Deep, whose accuracy drops as the inference speed goes up, PABEE has an inverted-U curve. We confirm this observation statistically with Monte Carlo simulation in Appendix 3.8. To analyze, when the patience t is set too large, the later internal classifier may suffer from the overthinking problem and make a wrong prediction that breaks the stable state among previous internal classifiers, which have not met the early-exit criterion because t is large. This makes PABEE leave more samples to be classified by the final classifier C_n, which suffers from the aforementioned overthinking problem. Thus, the effect of the Ensemble Learning vanishes and undermines its performance. Similarly, when t is relatively small, more samples may meet the early-exit criterion by accident before actually reaching the stable state where consecutive internal classifiers agree with each other.

Impact of Model Depth

We also investigate the impact of model depth on the performance of PABEE. We apply PABEE to a 24-layer ALBERT-large model. As shown in Table 3.4, our approach consistently improves the accuracy as more layers and classifiers are added while producing an even larger speed-up ratio. This finding demonstrates the potential of PABEE for burgeoning deeper PLMs [Shoeybi et al., 2019, Raffel et al., 2020, Brown et al., 2020].

Table 3.4. Experimental results (median of 5 runs) of different sizes of ALBERT on GLUE development set.

Method	**#Param**	**#Layer**	**Speed -up**	**MNLI** (393K)	**SST-2** (67K)	**STS-B** (5.7K)
ALBERT-base [Lan et al., 2020]	12M	12	1.00×	84.6	92.8	89.5
+ PABEE	12M	12	1.57×	**85.1**	**93.0**	**90.1**
ALBERT-large [Lan et al., 2020]	18M	24	1.00×	86.4	94.9	90.4
+ PABEE	18M	24	2.42×	**86.8**	**95.2**	**90.6**

Table 3.5. Results on the adversarial robustness. "Query Number" denotes the number of queries the attack system made to the target model and a higher number indicates more difficulty.

Metric	**ALBERT**			**+ Shallow-Deep [Kaya et al., 2019]**			**+ PABEE** *(ours)*		
(↑ better)	SNLI	MNLI-m/-mm	Yelp	SNLI	MNLI-m/-mm	Yelp	SNLI	MNLI-m/-mm	Yelp
Original Acc.	89.6	84.1 / 83.2	97.2	89.4	82.2 / 80.5	97.2	**89.9**	**85.0 / 84.8**	**97.4**
After-Attack Acc.	5.5	9.8 / 7.9	7.3	9.2	15.4 / 13.8	11.4	**19.3**	**30.2 / 25.6**	**18.1**
Query Number	58	80 / 86	841	64	82 / 86	870	**75**	**88 / 93**	**897**

3.3.6 Defending Against Adversarial Attack

Deep Learning models have been found to be vulnerable to adversarial examples that are slightly altered with perturbations often indistinguishable to humans [Kurakin et al., 2017]. Jin et al. [2020] revealed that PLMs can also be attacked with a high success rate. Recent studies [Kaya et al., 2019, Hu et al., 2020] attribute the vulnerability partially to the overthinking problem, arguing that it can be mitigated by early exit mechanisms.

In our experiments, we use a state-of-the-art adversarial attack method, TextFooler [Jin et al., 2020], which demonstrates effectiveness on attacking BERT. We conduct black-box attacks on three datasets: SNLI [Bowman et al., 2015], MNLI [Williams et al., 2018] and Yelp [Zhang et al., 2015]. Note that since we use the pre-tokenized data provided by Jin et al. [2020], the results on MNLI differ slightly from the ones in Table 3.1. We attack the original ALBERT-base model, ALBERT-base with Shallow-Deep [Kaya et al., 2019] and with Patience-based Early Exit.

As shown in Table 3.5, we report the original accuracy, after-attack accuracy and the number of queries needed by TextFooler to attack each model. Our approach successfully defends more than 3× attacks compared to the original ALBERT on NLI tasks, and 2× on the Yelp sentiment analysis task. Also, PABEE increases the number of queries needed to attack by a large margin, providing more protection to the model. Compared to Shallow-Deep [Kaya et al., 2019], our model demonstrates significant robustness improvements. To analyze, although the early exit mechanism of Shallow-Deep can prevent the aforementioned overthinking problem, it still relies on a single classifier to make the final prediction, which makes it vulnerable to adversarial attacks. In comparison, since Patience-based Early Exit exploits multiple layers and classifiers, the attacker has to fool multiple classifiers (which may exploit different features) at the same time, making it much more difficult to attack the model. This effect is similar to the merits of Ensemble Learning against adversarial attack, discussed in previous studies [Strauss et al., 2017, Tramèr et al., 2018, Pang et al., 2019].

3.4 Discussion

In this chapter, we proposed PABEE, a novel efficient inference method that can yield better accuracy-speed trade-off than existing methods. We verify its effectiveness and efficiency on GLUE and provide theoretical analysis. Empirical results show that PABEE can simultaneously improve the efficiency, accuracy, and adversarial robustness upon a competitive ALBERT model. However, some limitations should be noted. First, PABEE requires a relative deep model to effectively apply the patience mechanism, making it inapplicable for shallow models. Second, PABEE cannot work on multi-branch networks (e.g., NASNet [Zoph et al., 2018]) but only models with a single branch (e.g., ResNet, Transformer). For future work, we would like to explore our method on more tasks and settings. Also, since PABEE is orthogonal to prediction distribution based early exit approaches, it would be interesting to see if we can combine them with PABEE for better performance.

3.5 Broader Impact

As an efficient inference technique, our proposed PABEE can facilitate more applications on mobile and edge computing, and also help reduce energy use and carbon emission [Schwartz et al., 2020a]. Since our method serves as a plug-in for existing pretrained language models, it does not introduce significant new ethical concerns but more work is needed to determine its effect on biases (e.g., gender bias) that have already been encoded in a PLM.

3.6 Additional Results on Image Classification

To verify the effectiveness of PABEE on Computer Vision, we follow the experimental settings in Shallow-Deep [Kaya et al., 2019], we conduct experiments on two image classification datasets, CIFAR-10 and CIFAR-100 [Krizhevsky et al., 2009]. We use ResNet-56 [He et al., 2016] as the backbone and compare PABEE with BranchyNet [Teerapittayanon et al., 2016] and Shallow-Deep [Kaya et al., 2019]. After every two convolutional layers, an internal classifier is added. We set the batch size to 128 and use SGD optimizer with learning rate of 0.1.

Table 3.6. Experimental results (median of 5 runs) of ResNet based models on CIFAR-10 and CIFAR-100 datasets.

Method	**CIFAR-10**		**CIFAR-100**	
	Speed-up	Acc.	Speed-up	Acc.
ResNet-56 [He et al., 2016]	1.00×	91.8	1.00×	68.6
BranchyNet [Teerapittayanon et al., 2016]	1.33×	91.4	1.29×	68.2
Shallow-Deep [Kaya et al., 2019]	1.35×	91.6	1.32×	68.8
PABEE *(ours)*	1.26×	**92.0**	1.22×	**69.1**

The experimental results in CIFAR are reported in Table 3.6. PABEE outperform the original ResNet model by 0.2 and 0.5 in terms of accuracy while speed up the inference by 1.26× and 1.22× on CIFAR-10 and CIFAR-100, respectively. Also, PABEE demonstrates a better performance and a similar speed-up ratio compared to both baselines.

3.7 Proof of Theorem 1

Proof 3.7.1 *Recap we are in the case of binary classification. We denote the patience of PABEE as t, the total number of internal classifiers (IC) as n, the misclassification probability (i.e., error rate) of all internal classifiers as q, and the misclassification probability of the final classifier and the original classifier as p. We want to prove the PABEE mechanism improves the accuracy of conventional inference as long as* $n - t < (\frac{1}{2q})^{t+1} p - q$.

For the examples that do not early-stopped, the misclassification probability with and without PABEE is the same. Therefore, we only need to consider the ratio between the probability that a sample is early-stopped and misclassified (denoted as p_{misc}*) and that a sample is early-stopped (denoted as* p_{stop}*). We want to find the condition on n and t which makes* $\frac{p_{misc}}{p_{stop}} < p$.

First, considering only the probability mass of the model consecutively output the same label from the first position, we have

$$p_{stop} > q^{t+1} + (1-q)^{t+1} \tag{3.7}$$

which is the lower bound of p_{stop} *that only considering the probability of a sample is early-stopped by consecutively predicted to be the same label from the first internal classifier. We then take its derivative and find it obtains its minimum when* $q = 0.5$*. This corresponds to the case where the classification is performing random guessing (i.e. classification probability for class 0 and 1 equal to 0.5). Intuitively, in the random guessing case the internal classification results are most instable so the probability that a sample is early-stopped is the smallest.*

Therefore, we have $p_{stop} > (\frac{1}{2})^t$.

Then for p_{misc}*, we have*

$$p_{misc} < q^{t+1} + (n-t-1)(1-q)q^{t+1} \tag{3.8}$$

where q^{t+1} *is the probability that the example is consecutively misclassified for t+1 times* *from the first IC. The term* $(n-t-1)(1-q)q^{t+1}$ *is the summation of probability that the example is consecutively misclassified for t+1 times from the* $2, ..., n-t$ *th IC but correctly classified at the previous IC, without considering the cases that the the inference may already finished (whether correctly or not) before that IC. The summation of these two terms is an upper bound of* p_{misc}.

So we need to have

$$(n-t)q^{t+1} - (n-m-1)q^{t+2} < (\frac{1}{2})^t p \tag{3.9}$$

which equals to

$$(n-t)(q^t - q^{t+1}) < (\frac{1}{2})^t (\frac{p}{q}) - q^{t+1} \tag{3.10}$$

which equals to

$$n-t < \frac{(\frac{1}{2q})^t (\frac{p}{q}) - q}{1-q} < (\frac{1}{2q})^t (\frac{p}{q}) - q \tag{3.11}$$

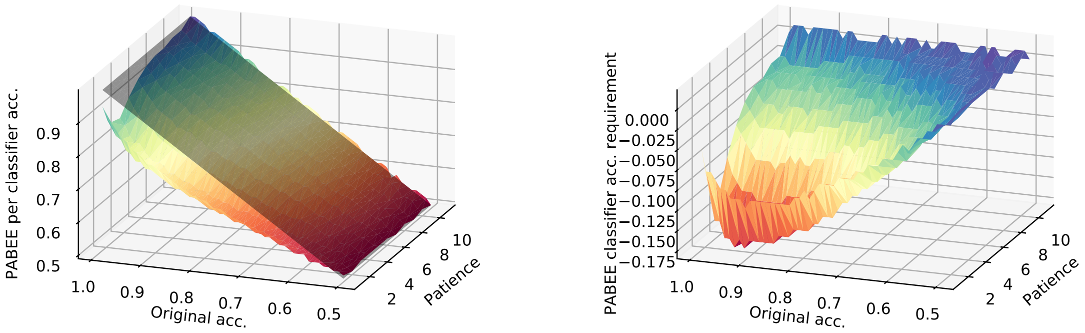
■

Specially, when $q = p$, the condition becomes $n-t < (\frac{1}{2p})^t - p$

3.8 Additional Results of Monte Carlo Simulation

To verify the theoretical feasibility of Patience-based Early Exit, we conduct Monte Carlo simulation. We simplify the task to be a binary classification with a 12-layer model which has classifiers $C_1 \dots C_{12}$ that all have the same probability to correctly predict.

Shown in Figure 3.5a, we illustrate the accuracy lower bound of each single C_i needed for PABEE to reach the same accuracy as the original accuracy without PABEE. We run the simulation 10,000 times with random Bernoulli Distribution sampling for every 0.01 of the original accuracy between 0.5 and 1.0 with patience $t \in [1, 11]$. The result shows that Patience-based Early Exit can effectively reduce the needed accuracy for each classifier. Additionally,



(a) Accuracy lower bound of each single PABEE classifier to achieve the original accuracy. The translucent black plain denotes inference without PABEE.

(b) Accuracy requirement reduction effect of PABEE classifiers.

Figure 3.5. Monte Carlo simulation of per PABEE classifier's accuracy vs. the original inference accuracy under different patience settings.

we illustrate the accuracy requirement reduction in Figure 3.5b. We can see a valley along the patience axis which matches our observation in Section 3.3.5. However, the best patience in favor of accuracy in our simulation is around 3 while in our experiments on real models and data suggest a patience setting of 6. To analyze, in the simulation we assume all classifiers have the same accuracy while in reality the accuracy is monotonically increasing with more layers involved in the calculation.

This chapter, in part, is a reprint of the material as it appears in “BERT Loses Patience: Fast and Robust Inference with Early Exit” by Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei, which was published at *Conference on Neural Information Processing Systems*, 2020. The dissertation author was the co-primary investigator and co-first author of this paper.

Chapter 4

Beyond Preserved Accuracy: Evaluating Loyalty and Robustness of BERT Compression

Recent studies on compression of pretrained language models (e.g., BERT) usually use preserved accuracy as the metric for evaluation. In this chapter, we propose two new metrics, label loyalty and probability loyalty that measure how closely a compressed model (i.e., student) mimics the original model (i.e., teacher). We also explore the effect of compression with regard to robustness under adversarial attacks. We benchmark quantization, pruning, knowledge distillation and progressive module replacing with loyalty and robustness. By combining multiple compression techniques, we provide a practical strategy to achieve better accuracy, loyalty and robustness.[1]

4.1 Introduction

Recently, many large pretrained language models (PLMs, Devlin et al., 2019, Liu et al., 2019b, Yang et al., 2019, Shoeybi et al., 2019, Raffel et al., 2020) have been proposed for a variety of Natural Language Processing (NLP) tasks. However, as pointed out in recent studies [Strubell et al., 2019, Schwartz et al., 2020a, Bender et al., 2021], these models suffer from computational inefficiency and high ecological cost. Many attempts have been made to address this problem,

---

[1] Our code is available at https://github.com/JetRunner/beyond-preserved-accuracy.

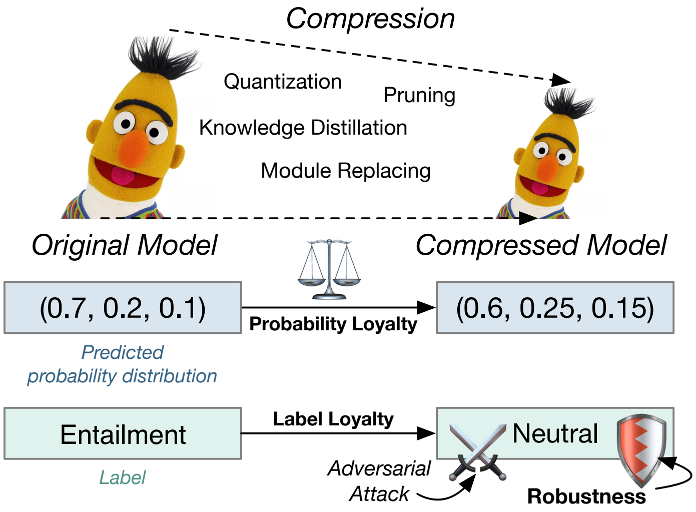


Figure 4.1. Three metrics to evaluate the compressed models beyond preserved accuracy. For each input, **label and probability loyalty** measure the shift of label and predicted probability distribution, respectively. **Robustness** measures the performance of the compressed model under adversarial attacks.

including quantization [Zafrir et al., 2019, Shen et al., 2020], pruning [Michel et al., 2019, Sanh et al., 2020], knowledge distillation (KD) [Sanh et al., 2019, Sun et al., 2019a, 2020b, Turc et al., 2019, Jiao et al., 2020, Wang et al., 2020, Zhou et al., 2022] and progressive module replacing [Xu et al., 2020].

BERT [Devlin et al., 2019] is a representative PLM. Many works compressing BERT use preserved accuracy with computational complexity (e.g., speed-up ratio, FLOPS, number of parameters) as metrics to evaluate compression. This evaluation scheme is far from perfect: (1) Preserved accuracy cannot reflect how alike the teacher and student[2] models behave. This can be problematic when applying compression techniques in production (to be detailed in Section 4.3). (2) Using preserved accuracy to evaluate models compressed with more data or data augmentation [Jiao et al., 2020] can be misleading, since one cannot tell whether the improvement should be attributed to the innovation of the compression technique or addition of data. (3) Model robustness, which is critical for production, is often missing from evaluation,

---

[2] Teacher and student are originally concepts in knowledge distillation. In this chapter, we will sometimes use teacher and student to refer to the original model and compressed model in other compression methods for simplicity.

leaving a possible safety risk.

As illustrated in Figure 4.1, to measure the resemblance between the student and teacher models, we propose *label loyalty* and *probability loyalty* to target different but important aspects. We also explore the robustness of the compressed models by conducting black-box adversarial attacks. We apply representative BERT compression methods of different types to the same teacher model and benchmark their performance in terms of accuracy, speed, loyalty and robustness. We find that methods with a knowledge distillation loss perform well on loyalty and that post-training quantization can drastically improve robustness against adversarial attacks. We use the conclusions drawn from these experiments to combine multiple techniques together and achieve significant improvement in terms of accuracy, loyalty and robustness.

4.2 BERT Compression

Compressing and accelerating pretrained language models like BERT has been an active field of research. Some initial work employs conventional methods for neural network compression to compress BERT. For example, Q8-BERT [Zafrir et al., 2019] and Q-BERT [Shen et al., 2020] employ weight quantization to reduce the number of bits used to represent a parameter in a BERT model. Pruning methods like Head Prune [Michel et al., 2019] and Movement Pruning [Sanh et al., 2020] remove weights based on their importance to reduce the memory footprint of pretrained models. Another line of research focuses on exploiting the knowledge encoded in a large pretrained model to improve the training of more compact models. For instance, DistilBERT [Sanh et al., 2019] and BERT-PKD [Sun et al., 2019a] employ knowledge distillation [Hinton et al., 2015] to train compact BERT models in a task-specific and task-agnostic fashion respectively by mimicking the behavior of large teacher models. Recently, Xu et al. [2020] proposed progressive module replacing, which trains a compact student model by progressively replacing the teacher layers with their more compact substitutes.

Table 4.1. Accuracy, loyalty and robustness of compressed models on the test set of MNLI (3 runs). Accuracy scores are from the GLUE [Wang et al., 2019b] test server. †These models are not initialized from (a part of) the finetuned BERT teacher. ‡The speed-up ratio of quantization is benchmarked on CPU. ↑Higher is better.

Method	# Layer	Speed-up ↑	MNLI m/mm ↑	Loyalty Label ↑	Loyalty Probability ↑	Adversarial Attack Acc ↑	Adversarial Attack #Query ↑
Teacher [Devlin et al., 2019]	12	1.0×	84.5 / 83.3	100	100	8.1(±0.1)	89.6(±0.1)
Truncate & Finetune	6	2.0×	81.1 / 80.0	87.7(±0.2)	84.9(±0.7)	4.4(±0.1)	78.0(±0.1)
Pure KD	6	2.0×	81.1 / 80.8	89.2(±0.1)	89.5(±0.2)	6.2(±0.1)	80.1(±0.2)
Q8-PTQ [Zafrir et al., 2019]	12	1.8×‡	80.7 / 80.4	89.6(±0.5)	80.8(±0.4)	40.2(±0.1)	91.6(±0.1)
Q8-QAT† [Zafrir et al., 2019]	12	1.8×‡	83.4 / 82.4	89.7(±0.2)	88.2(±0.3)	6.8(±0.2)	82.7(±0.2)
Head Prune [Michel et al., 2019]	12	1.2×	80.9 / 80.6	87.8(±0.1)	85.5(±0.6)	9.1(±0.1)	90.5(±0.2)
DistilBERT† [Sanh et al., 2019]	6	2.0×	82.4 / 81.4	88.9(±0.2)	88.4(±0.4)	5.9(±0.1)	80.8(±0.2)
TinyBERT† [Jiao et al., 2020]	6	2.0×	82.7 / 82.7	88.9(±0.1)	88.4(±0.7)	6.7(±0.1)	82.1(±0.2)
BERT-PKD [Sun et al., 2019a]	6	2.0×	81.3 / 81.1	88.9(±0.1)	89.0(±0.2)	6.4(±0.2)	81.9(±0.2)
BERT-of-Theseus [Xu et al., 2020]	6	2.0×	81.8 / 80.7	88.1(±0.2)	82.5(±0.3)	8.3(±0.2)	89.7(±0.2)

4.3 Metrics Beyond Accuracy

4.3.1 Loyalty

Label Loyalty

Model compression is a common practice to optimize the efficiency of a model for deployment [Cheng et al., 2017]. In real-world settings, training and deployment are often separate [Paleyes et al., 2020]. As such it is desirable to have a metric to measure to what extent the "production model" is different from the "development model". Moreover, when discussing ethical concerns, previous studies [Wang et al., 2020, Zhou et al., 2020] ignore the risk that model compression could introduce additional biases. However, a recent work [Hooker et al., 2020] strongly contradicts this assumption. In a nutshell, we would desire the student to behave as closely as possible to the teacher, to make it more predictable and minimize the risk of introducing extra bias. Label loyalty directly reflects the resemblance of the labels predicted between the teacher and student models. It is calculated in the same way as accuracy, but between the student's prediction and the teacher's prediction, instead of ground labels:

$$L_l = \text{Accuracy}(pred_t, pred_s) \tag{4.1}$$

where $pred_t$ and $pred_s$ are the predictions of the teacher and student, respectively.

Probability Loyalty

Except for the label correspondence, we argue that the predicted probability distribution matters as well. In industrial applications, calibration [Guo et al., 2017, Li et al., 2020a], which focuses on the meaningfulness of confidence, is an important issue for deployment. Many dynamic inference acceleration methods [Xin et al., 2020b, Schwartz et al., 2020b, Liu et al., 2020b, Xin et al., 2020a, Li et al., 2020a] use entropy or the maximum value of the predicted probability distribution as the signal for early exiting. Thus, a shift of predicted probability distribution in a compressed model could break the calibration and invalidate calibrated early exiting pipelines.

Kullback–Leibler (KL) divergence is often used to measure how one probability distribution is different from a reference distribution.

$$D_{\text{KL}}(P \| Q) = \sum_{x \in \mathscr{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{4.2}$$

where $\mathscr{X}$ is the probability space; P and Q are predicted probability distributions of the teacher and students, respectively. Here, we use its variant, the Jensen–Shannon (JS) divergence, since it is symmetric and always has a finite value which is desirable for a distance-like metric:

$$D_{\text{JS}}(P \| Q) = \frac{1}{2} D_{\text{KL}}(P \| M) + \frac{1}{2} D_{\text{KL}}(Q \| M) \tag{4.3}$$

where $M = \frac{1}{2}(P+Q)$. Finally, the probability loyalty between P and Q is defined as:

$$L_p(P\|Q) = 1 - \sqrt{D_{\mathrm{JS}}(P\|Q)} \tag{4.4}$$

where $L_p \in [0,1]$; higher L_p represents higher resemblance. Note that Equation 4.2 is also known as the KD loss [Hinton et al., 2015], thus KD-based methods will naturally have an advantage in terms of probability loyalty.

4.3.2 Robustness

Deep Learning models have been shown to be vulnerable to adversarial examples that are slightly altered with perturbations often indistinguishable to humans [Kurakin et al., 2017]. Previous work [Su et al., 2018] found that small convolutional neural networks (CNN) are more vulnerable to adversarial attacks compared to bigger ones. Likewise, we intend to investigate how BERT models perform and the effect of different types of compression in terms of robustness. We use an off-the-shelf adversarial attack method, TextFooler [Jin et al., 2020], which demonstrates state-of-the-art performance on attacking BERT. TextFooler conducts black-box attacks by querying the BERT model with the adversarial input where words are perturbed based on their part-of-speech role. We select two metrics from [Jin et al., 2020], after-attack accuracy and query number, to evaluate a model's robustness. After-attack accuracy represents the remaining accuracy after the adversarial attack. Query number represents how many queries with perturbed input have been made to complete the attack.

4.4 Experiments

4.4.1 Dataset and Baselines

We use the Multi-Genre Natural Language Inference dataset (MNLI) [Williams et al., 2018] for evaluation. The accuracy scores are from the GLUE benchmark [Wang et al., 2019b] test server. We select representative methods of different types of compression methods: Head

Prune [Michel et al., 2019] for pruning; Post-training Quantization (PTQ) and Quantization-aware Training (QAT) [Zafrir et al., 2019] for quantization; DistilBERT [Sanh et al., 2019] and TinyBERT [Jiao et al., 2020] for pretraining-phase knowledge distillation; BERT-PKD [Sun et al., 2019a] for downstream knowledge distillation; and BERT-of-Theseus [Xu et al., 2020] for module replacing. Following Sun et al. [2019a], Xu et al. [2020], we truncate the first (bottom) 6 layers and then finetune it as a baseline for 6-layer models. Additionally, we directly optimize the KL divergence (i.e., pure KD loss) to set an upper bound for probability loyalty.

4.4.2 Training Details

Our implementation is based on Hugging Face Transformers [Wolf et al., 2020]. We first finetune a BERT-base model to be the teacher for KD and the source model for quantization and pruning. The learning rate is set to 3×10^{-5} and the batch size is 64 with 1,000 warm-up steps. For quantization and pruning, the source model is the same finetuned teacher. For downstream KD and BERT-of-Theseus, we initialize the model by truncating the first (bottom) 6 layers of the finetuned teacher, following the original papers [Sun et al., 2019a, Xu et al., 2020]. QAT uses pretrained BERT-base for initialization. For pretraining distillation, we directly finetune compressed 6-layer DistilBERT and TinyBERT checkpoints to report results. The pruning percentage for Head Prune is 45%. The hyperparameters of BERT-PKD are from the original implementation.

4.4.3 Experimental Results

We show experimental results in Table 4.1. First, we find that post-training quantization can drastically improve model robustness. A possible explanation is that the regularization effect of post-training quantization [Paupamah et al., 2020, Wu and Flierl, 2020] helps improve the robustness of the model [Werpachowski et al., 2019, Ma et al., 2020]. A similar but smaller effect can be found from pruning. However, as shown in Table 4.2, if we finetune the low-precision or pruned model again, the model would re-overfit the data and yield even lower robustness

Table 4.2. Accuracy and loyalty for combining multiple compression techniques on the test set of MNLI. **L-L**: label loyalty; **P-L**: probability loyalty; **AA**: after-attack accuracy; **# Q**: Query number for adversarial attack. The number of layers for each group is consistent with Table 4.1.

Method	Speed	MNLI	L-L	P-L	AA	# Q
Teacher	1.0×	84.5 / 83.3	100	100	8.1	89.6
Head Prune	1.2×	80.9 / 80.6	87.8	85.5	9.1	90.5
+Finetune	1.2×	83.2 / 81.9	89.1	85.5	7.2	83.2
+KD	1.2×	**84.2 / 83.0**	**93.3**	**93.0**	8.3	90.5
+KD+PTQ	2.2×	80.8 / 80.4	89.6	86.3	**38.4**	**90.9**
Q8-QAT	1.8×	83.4 / 82.4	89.7	88.2	6.8	82.7
Q8-PTQ	1.8×	80.7 / 80.4	89.6	80.8	**40.2**	**91.6**
+Finetune	1.8×	82.9 / 81.9	89.7	84.8	7.1	84.5
+KD	1.8×	**84.1 / 83.5**	**94.0**	**93.9**	7.5	86.1
BERT-PKD	2.0×	81.3 / 81.1	88.9	89.0	6.4	81.9
Theseus	2.0×	81.8 / 80.7	88.1	82.5	8.3	89.7
+KD	2.0×	**82.6 / 81.7**	**91.2**	**91.4**	8.0	88.7
+KD+PTQ	3.6×	80.2 / 79.9	89.5	80.3	**36.5**	**91.3**

than the original model. Second, KD-based models maintains good label loyalty and probability loyalty due to their optimization objectives. Interestingly, compared to *Pure KD* where we directly optimize the KL divergence, DistilBERT, TinyBERT and BERT-PKD trade some loyalty in exchange for accuracy. Compared to DistilBERT, TinyBERT digs up higher accuracy by introducing layer-to-layer distillation, with their loyalty remains identical. Also, we do not observe a significant difference between pretraining KD and downstream KD in terms of both loyalty and robustness ($p > 0.1$). Notably, BERT-of-Theseus has a significantly lower loyalty, suggesting the mechanism behind it is different from KD. We also provide some results on SST-2 [Socher et al., 2013] in Section 4.7.

4.5 Combining the Bag of Tricks

As we described in Section 4.4.3, we discover that post-training quantization (PTQ) can improve the robustness of a model while knowledge distillation (KD) loss benefits the loyalty of a compressed model. Thus, by combining multiple compression techniques, we expect to achieve a higher speed-up ratio with improved accuracy, loyalty and robustness.

To combine KD with other methods, we replace the original cross-entropy loss in quantization-aware training and module replacing with the knowledge distillation loss [Hinton et al., 2015] as in Equation 4.2. For pruning, we perform knowledge distillation on the pruned model. We also apply the temperature re-scaling trick from [Hinton et al., 2015] with a fixed temperature of 10. As shown in Table 4.2, the knowledge distillation loss effectively improves the accuracy and loyalty of pruning, quantization and module replacing.

Furthermore, we post-quantize the KD-enhanced models after they are trained. Shown in Table 4.2, by adding post-training quantization, the speed and robustness can both be boosted. Notably, the order to apply PTQ and KD does matter. PTQ→KD has high accuracy and loyalty but poor robustness while KD→PTQ remains a good robustness with a lower accuracy performance. To summarize, we recommend the following compression strategy: (1) conduct pruning or module replacing with a KD loss; (2) for speed-sensitive and robustness-sensitive applications, apply post-training quantization afterwards.

4.6 Conclusion

In this chapter, we propose label and probability loyalty to measure the correspondence of label and predicted probability distribution between compressed and original models. In addition to loyalty, we investigate the robustness of different compression techniques under adversarial attacks. These metrics reveal that post-training quantization and knowledge distillation can drastically improve robustness and loyalty, respectively. By combining multiple compression methods, we can further improve speed, accuracy, loyalty and robustness for various applications. Our metrics help mitigate the gap between model training and deployment, shed light upon comprehensive evaluation for compression of pretrained language models, and call for the invention of new compression techniques.

4.7 Additional Experimental Results on SST-2

Table 4.3. Accuracy and loyalty of some compression techniques on the test set of SST-2. **L-L**: label loyalty; **P-L**: probability loyalty; **AA**: after-attack accuracy; **# Q**: Query number for adversarial attack. The number of layers for each group is consistent with Table 4.1.

Method	Speed	SST-2	L-L	P-L	AA	# Q
Teacher	1.0×	92.0	100	100	7.5	81.2
KD	2.0×	**91.5**	**93.8**	**92.4**	7.2	80.3
Head Prune	1.3×	90.4	89.5	88.2	8.1	81.0
Q8-QAT	1.8×	91.4	91.8	90.3	6.6	81.4
Q8-PTQ	1.8×	90.1	91.3	88.9	**26.5**	**86.4**

This chapter, in part, is a reprint of the material as it appears in “Beyond Preserved Accuracy: Evaluating Loyalty and Robustness of BERT Compression” by Canwen Xu, Wangchunshu Zhou, Tao Ge, Ke Xu, Julian McAuley, and Furu Wei, which was published at *Empirical Methods in Natural Language Processing*, 2021. The dissertation author was the co-primary investigator and co-first author of this paper.

Chapter 5

BERT Learns to Teach: Knowledge Distillation with Meta Learning

In this chapter, we present Knowledge Distillation with Meta Learning (MetaDistil), a simple yet effective alternative to traditional knowledge distillation (KD) methods where the teacher model is fixed during training. We show the teacher network can learn to better transfer knowledge to the student network (i.e., *learning to teach*) with the feedback from the performance of the distilled student network in a meta learning framework. Moreover, we introduce a pilot update mechanism to improve the alignment between the inner-learner and meta-learner in meta learning algorithms that focus on an improved inner-learner. Experiments on various benchmarks show that MetaDistil can yield significant improvements compared with traditional KD algorithms and is less sensitive to the choice of different student capacity and hyperparameters, facilitating the use of KD on different tasks and models.[1]

5.1 Introduction

With the prevalence of large neural networks with millions or billions of parameters, model compression is gaining prominence for facilitating efficient, eco-friendly deployment for machine learning applications. Among techniques for compression, knowledge distillation (KD) [Hinton et al., 2015] has shown effectiveness in both Computer Vision and Natural

---

[1] The code is available at https://github.com/JetRunner/MetaDistil.

Language Processing tasks [Hinton et al., 2015, Romero et al., 2015, Zagoruyko and Komodakis, 2017, Tung and Mori, 2019, Peng et al., 2019, Ahn et al., 2019, Park et al., 2019, Passalis and Tefas, 2018, Heo et al., 2019, Kim et al., 2018, Shi et al., 2021, Sanh et al., 2019, Jiao et al., 2020, Wang et al., 2020]. Previous works often train a large model as the "teacher"; then they fix the teacher and train a "student" model to mimic the behavior of the teacher, in order to transfer the knowledge from the teacher to the student.

However, this paradigm has the following drawbacks: **(1) The teacher is unaware of the student's capacity.** Recent studies in pedagogy suggest student-centered learning, which considers students' characteristics and learning capability, has shown effectiveness improving students' performance [Cornelius-White, 2007, Wright, 2011]. However, in conventional knowledge distillation, the student passively accepts knowledge from the teacher, without regard for the student model's learning capability and performance. Recent works [Park et al., 2021, Shi et al., 2021] introduce student-aware distillation by jointly training the teacher and the student with task-specific objectives. However, there is still space for improvement since: **(2) The teacher is not optimized for distillation.** In previous works, the teacher is often trained to optimize its *own* inference performance. However, the teacher is not aware of the need to transfer its knowledge to a student and thus usually does so suboptimally. A real-world analogy is that a PhD student may have enough knowledge to solve problems themselves, but requires additional teaching training to qualify as a professor.

To address these two drawbacks, we propose Knowledge Distillation with Meta Learning (MetaDistil), a new teacher-student distillation framework using meta learning [Finn et al., 2017] to exploit feedback about the student's learning progress to improve the teacher's knowledge transfer ability throughout the distillation process. On the basis of previous formulations of bi-level optimization based meta learning [Finn et al., 2017], we propose a new mechanism called *pilot update* that aligns the learning of the bi-level learners (i.e., the teacher and the student). We illustrate the workflow of MetaDistil in Figure 5.1. The teacher in MetaDistil is trainable, which enables the teacher to adjust to its student network and also improves its "teaching skills."

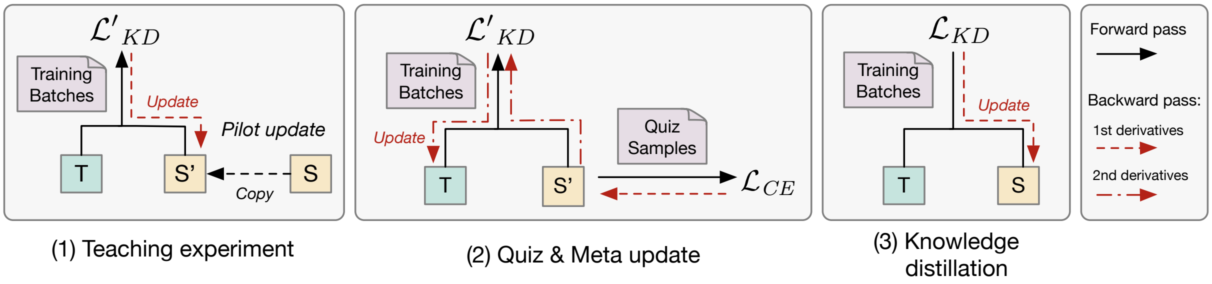


Figure 5.1. The workflow of MetaDistil. (1) We perform experimental knowledge distillation on a selection of training batches. Instead of updating the student S, we make a temporary copy S' and update S'. (2) We calculate a Cross-Entropy loss $\mathscr{L}_{CE}$ of S' on samples from a separate quiz set. We calculate the gradients of $\mathscr{L}_{CE}$ with respect to the parameters of T and update T by gradient descent. (3) We discard S' and use the updated T to perform actual knowledge distillation and update S.

Motivated by the idea of student-centered learning, we allow the teacher to adjust its output based on the performance of the student model on a "quiz set," which is a separate reserved data split from the original training set. For each training step, we first copy the student S to S' and update S' by a common knowledge distillation loss. We call this process a "teaching experiment." In this way, we can obtain an experimental student S' that can be quizzed. Then, we sample from the quiz set, and calculate the loss of S' on these samples. We use this loss as a feedback signal to meta-update the teacher by calculating second derivatives and performing gradient descent [Finn et al., 2017]. Finally, we discard the experimental subject S' and use the updated teacher to distill into the student S on the same training batches. The use of meta learning allows the teacher model to receive feedback from the student in a completely differentiable way. We provide a simple and intuitive approach to explicitly optimize the teacher using the student's quiz performance as a proxy.

To test the effectiveness of MetaDistil, we conduct extensive experiments on text and image classification tasks. MetaDistil outperforms knowledge distillation by a large margin, verifying the effectiveness and versatility of our method. Also, our method achieves state-of-the-art performance compressing BERT [Devlin et al., 2019] on the GLUE benchmark [Wang et al., 2019b] and shows competitive results compressing ResNet [He et al., 2016] and VGG [Si-

Algorithm 1. Knowledge Distillation with Meta Learning (MetaDistil)

```
Require: student $\theta_S$, teacher $\theta_T$, train set $\mathscr{D}$, quiz set $\mathscr{Q}$
Require: $\lambda$, $\mu$: learning rate for the student and the teacher
while not done do
    Sample batch of training data $\boldsymbol{x} \sim \mathscr{D}$
    Copy student parameter $\theta_S$ to student $\theta'_S$
    Update $\theta'_S$ with $\boldsymbol{x}$ and $\theta_T$: $\theta'_S \leftarrow \theta'_S - \lambda \nabla_{\theta'_S} \mathscr{L}_S(x; \theta_S; \theta_T)$
    Sample a batch of quiz data $\boldsymbol{q} \sim \mathscr{Q}$
    Update $\theta_T$ with $\boldsymbol{q}$ and $\theta'_S$: $\theta_T \leftarrow \theta_T - \mu \nabla_{\theta_T} \mathscr{L}_{\mathscr{T}}(\boldsymbol{q}, \theta'_S(\theta_T))$
    Update original $\theta_S$ with $\boldsymbol{x}$ and the updated $\theta_T$: $\theta_S \leftarrow \theta_S - \lambda \nabla_{\theta_S} \mathscr{L}_S(x; \theta_S; \theta_T)$
end while
```

monyan and Zisserman, 2015] on CIFAR-100 [Krizhevsky et al., 2009]. Additionally, we design experiments to analyze and explain the improvement. Ablation studies show the effectiveness of our proposed pilot update and dynamic distillation. Also, compared to conventional KD, MetaDistil is more robust to different student capacity and hyperparameters, which is probably because of its ability to adjust the parameters of the teacher model.

5.2 Knowledge Distillation with Meta Learning

An overview of MetaDistil is presented in Figure 5.1. MetaDistil includes two major components. First, the meta update enables the teacher model to receive the student model's feedback on the distillation process, allowing the teacher model to "learn to teach" and provide distillation signals that are more suitable for the student model's current capacity. The pilot update mechanism ensures a finer-grained match between the student model and the meta-updated teacher model.

5.2.1 Background

Knowledge Distillation

Knowledge distillation algorithms aim to exploit the hidden knowledge from a large teacher network, denoted as T, to guide the training of a shallow student network, denoted as S. To help transfer the knowledge from the teacher to the student, apart from the original task-specific objective (e.g., cross-entropy loss), a knowledge distillation objective which aligns the behavior of the student and the teacher is included to train the student network. Formally, given a labeled dataset $\mathscr{D}$ of N samples $\mathscr{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, we can write the loss function of the student network as follows,

$$\begin{aligned} \mathscr{L}_S(\mathscr{D}; \theta_S; \theta_T) = \frac{1}{N} \sum_{i=1}^{N} [\alpha \mathscr{L}_{\mathscr{T}}(y_i, S(x_i; \theta_S)) \\ + (1-\alpha) \mathscr{L}_{KD}(T(x_i; \theta_T), S(x_i; \theta_S))] \end{aligned} \tag{5.1}$$

where α is a hyper-parameter to control the relative importance of the two terms; θ_T and θ_S are the parameters of the teacher T and student S, respectively. $\mathscr{L}_{\mathscr{T}}$ refers to the task-specific loss and $\mathscr{L}_{KD}$ refers to the knowledge distillation loss which measures the similarity of the student and the teacher. Some popular similarity measurements include the KL divergence between the output probability distribution, the mean squared error (MSE) between student and teacher logits, the similarity between the student and the teacher's attention distribution, etc. We do not specify the detailed form of the loss function because MetaDistil is a general framework that can be easily applied to various kinds of KD objectives as long as the objective is differentiable with respect to the teacher parameters. In the experiments of this chapter, we use mean squared error between the hidden states of the teacher and the student for both our method and the KD baseline since recent study Kim et al. [2021] finds that it is more stable and slightly outperforms than KL divergence.

Meta Learning

In meta learning algorithms that involve a bi-level optimization problem [Finn et al., 2017], there exists an inner-learner f_i and a meta-learner f_m. The inner-learner is trained to accomplish a task $\mathscr{T}$ or a distribution of tasks with help from the meta-learner. The training process of f_i on $\mathscr{T}$ with the help of f_m is typically called *inner-loop*, and we can denote $f_i'(f_m)$ as the updated inner-learner after the inner-loop. We can express f_i' as a function of f_m because learning f_i depends on f_m. In return, the meta-learner is optimized with a meta objective, which is generally the maximization of expected performance of the inner-learner after the inner-loop, i.e., $f_i'(f_m)$. This learning process is called a *meta-loop* and is often accomplished by gradient descent with derivatives of $\mathscr{L}(f_i'(f_m))$, the loss of updated inner-leaner on some held-out support set (i.e., the quiz set in this chapter).

5.2.2 Methodology

Pilot Update

In the original formulation of meta learning [Finn et al., 2017], the purpose is to learn a good meta-learner f_m that can generalize to different inner-learners f_i for different tasks. In their approach, the meta-learner is optimized for the "original" inner-learner at the beginning of each iteration and the current batch of training data. The updated meta-learner is then applied to the updated inner-learner and a different batch of data in the next iteration. This behavior is reasonable if the purpose is to optimize the meta-learner. However, in MetaDistil, we only care about the performance of the only inner-learner, i.e., the student. In this case, this behavior leads to a mismatch between the meta-learner and the inner-learner, and is therefore suboptimal for learning a good inner-learner. Therefore, we need a way to align and synchronize the learning of the meta- and inner-learner, in order to allow an update step of the meta-learner to have an instant effect on the inner-learner. This instant reflection prevents the meta-learner from catastrophic forgetting [McCloskey and Cohen, 1989]. To achieve this, we design a pilot update mechanism. For a batch of training data $\boldsymbol{x}$, we first make a temporary copy of the inner-learner f_i and update both the copy f_i' and the meta learner f_m on $\boldsymbol{x}$. Then, we discard f_i' and update f_i again with the updated f_m on the same data $\boldsymbol{x}$. This mechanism can apply the impact of data $\boldsymbol{x}$ to both f_m and f_i at the same time, thus aligns the training process. Pilot update is a general technique that can potentially be applied to any meta learning application that optimizes the inner-learner performance. We will describe how we apply this mechanism to MetaDistil shortly and empirically verify the effectiveness of pilot update in Section 5.3.2.

Learning to Teach

In MetaDistil, we would like to optimize the teacher model, which is fixed in traditional KD frameworks. Different from previous deep mutual learning [Zhang et al., 2018b] methods that switch the role between the student and teacher network and train the original teacher model with soft labels generated by the student model, or recent works [Shi et al., 2021, Park et al., 2021] that update the teacher model with a task-specific loss during the KD process, MetaDistil explicitly optimizes the teacher model in a "learning to teach" fashion, so that it can better transfer its knowledge to the student model. Concretely, the optimization objective of the teacher model in the MetaDistil framework is *the performance of the student model after distilling from the teacher model.* This "learning to teach" paradigm naturally fits the bi-level optimization framework in meta learning literature.

In the MetaDistil framework, the student network θ_S is the inner-learner and the teacher network θ_T is the meta-learner. For each training step, we first copy the student model θ_S to an "experimental student" θ'_S. Then given a batch of training examples $\boldsymbol{x}$ and the learning rate λ, the experimental student is updated in the same way as conventional KD algorithms:

$$\theta'_S(\theta_T) = \theta_S - \lambda \nabla_{\theta_S} \mathscr{L}_S(x; \theta_S; \theta_T). \tag{5.2}$$

To simplify notation, we will consider one gradient update for the rest of this section, but using multiple gradient updates is a straightforward extension. We observe that the updated experimental student parameter θ'_S, as well as the student quiz loss $l_q = \mathscr{L}_{\mathscr{T}}(\boldsymbol{q}, \theta'_S(\theta_T))$ on a batch of quiz samples $\boldsymbol{q}$ sampled from a held-out quiz set $\mathscr{Q}$, is a function of the teacher parameter θ_T. Therefore, we can optimize l_q with respect to θ_T by a learning rate μ:

$$\theta_T \leftarrow \theta_T - \mu \nabla_{\theta_T} \mathscr{L}_{\mathscr{T}} \left(\boldsymbol{q}, \theta'_S(\theta_T)\right) \tag{5.3}$$

We evaluate the performance of the experimental student on a separate quiz set to prevent overfitting the validation set, which is preserved for model selection. Note that the student is never trained on the quiz set and the teacher only performs meta-update on the quiz set instead of fitting it. We do not use a dynamic quiz set strategy because otherwise the student would have been trained on the quiz set and the loss would not be informative. After meta-updating the teacher model, we then update the "real" student model in the same way as described in Equation 5.2. Intuitively, optimizing the teacher network θ_T with Equation 5.3 is maximizing the expected performance of the student network after being taught by the teacher with the KD objective in the inner-loop. This meta-objective allows the teacher model to adjust its parameters to better transfer its knowledge to the student model. We apply the pilot update strategy described in Section 5.2.2 to better align the learning of the teacher and student, as shown in Algorithm 1.

5.3 Experiments

5.3.1 Experimental Setup

We evaluate MetaDistil on two commonly used classification benchmarks for knowledge distillation in both Natural Language Processing and Computer Vision (see Section 5.6).

Settings

For NLP, we evaluate our proposed approach on the GLUE benchmark [Wang et al., 2019b]. Specifically, we test on MRPC [Dolan and Brockett, 2005], QQP and STS-B [Conneau and Kiela, 2018] for Paraphrase Similarity Matching; SST-2 [Socher et al., 2013] for Sentiment Classification; MNLI [Williams et al., 2018], QNLI [Rajpurkar et al., 2016] and RTE [Wang et al., 2019b] for the Natural Language Inference; CoLA [Warstadt et al., 2019] for Linguistic Acceptability. Following previous studies [Sun et al., 2019a, Jiao et al., 2020, Xu et al., 2020], our goal is to distill BERT-Base [Devlin et al., 2019] into a 6-layer BERT with the hidden size of 768. We use MSE loss between model logits as the distillation objective. The reported results are in the same format as on the GLUE leaderboard. For MNLI, we report the results on MNLI-m and MNLI-mm, respectively. For MRPC and QQP, we report both F1 and accuracy. For STS-B,

Table 5.1. Experimental results on the development set and the test set of GLUE. Numbers under each dataset indicate the number of training samples. All student models have the same architecture of 66M parameters, 6 Transformer layers and 1.94× speed-up. The test results are from the official test server of GLUE. The best results for the task-specific setting are marked with **boldface**. Results reported by us are average of 3 runs with different seeds. †Results reported by us. The student is initialized with a 6-layer pretrained BERT [Turc et al., 2019] thus has a *better* performance than the original implementation. ‡TinyBERT has data augmentation (DA).

Method	**#Param.**	**Speed-up**	**GLUE** [Wang et al., 2019b]							
			CoLA (8.5K)	**MNLI** (393K)	**MRPC** (3.7K)	**QNLI** (105K)	**QQP** (364K)	**RTE** (2.5K)	**SST-2** (67K)	**STS-B** (5.7K)
Dev Set										
BERT-Base (teacher) [2019]	110M	1.00×	58.9	84.6/84.9	91.6/87.6	91.2	88.5/91.4	71.4	93.0	90.2/89.8
BERT-6L (student) [2019]	66M	1.94×	53.5	81.1/81.7	89.2/84.4	88.6	86.9/90.4	67.9	91.1	88.1/87.9
Pretraining Distillation										
TinyBERT‡ [2020]	66M	1.94×	54.0	84.5/84.5	90.6/86.3	91.1	88.0/91.1	73.4	93.0	90.1/89.6
MiniLM [2020]	66M	1.94×	49.2	84.0/ -	88.4/ -	91.0	- /91.0	71.5	92.0	-
MiniLM v2 [2021a]	66M	1.94×	52.5	84.2/ -	88.9/ -	90.8	- /91.1	72.1	92.4	-
Task-specific Distillation										
KD† [2015]	66M	1.94×	54.1	82.6/83.2	89.6/85.2	89.2	87.3/90.9	67.7	91.2	88.6/88.2
PKD† [2019a]	66M	1.94×	54.5	82.7/83.3	89.4/84.7	89.5	87.8/90.9	67.6	91.3	88.6/88.1
TinyBERT w/o DA†	66M	1.94×	52.4	**83.6/83.8**	90.5/86.5	89.8	87.6/90.6	67.7	91.9	89.2/88.7
RCO† [2019]	66M	1.94×	53.6	82.4/82.9	89.5/85.1	89.7	87.4/90.6	67.6	91.4	88.7/88.3
TAKD† [2020]	66M	1.94×	53.8	82.5/83.0	89.6/85.0	89.6	87.5/90.7	68.5	91.4	88.2/88.0
DML† [2018b]	66M	1.94×	53.7	82.4/82.9	89.6/85.1	89.6	87.4/90.3	68.4	91.5	88.4/88.1
ProKT† [2021]	66M	1.94×	54.3	82.8/83.2	90.7/86.3	89.7	87.9/90.9	68.4	91.3	88.9/88.6
SFTN† [2021]	66M	1.94×	53.6	82.4/82.9	89.8/85.3	89.5	87.5/90.4	68.5	91.5	88.4/88.5
MetaDistil *(ours)*	66M	1.94×	**58.6**	83.5/**83.8**	**91.1/86.8**	**90.4**	**88.1/91.0**	**69.4**	**92.3**	**89.4/89.1**
w/o pilot update	66M	1.94×	56.3	83.0/83.4	90.6/86.6	89.9	88.0/88.5	67.7	92.0	89.2/89.0
Test Set										
BERT-Base (teacher) [2019]	110M	1.00×	52.1	84.6/83.4	88.9/84.8	90.5	71.2/89.2	66.4	93.5	87.1/85.8
Pretraining Distillation										
DistilBERT [2019]	66M	1.94×	45.8	81.6/81.3	87.6/83.1	88.8	69.6/88.2	54.1	92.3	71.0/71.0
TinyBERT‡ [2020]	66M	1.94×	51.1	84.3/83.4	88.8/84.5	91.6	70.5/88.3	70.4	92.6	86.2/84.8
Task-specific Distillation										
KD [2019]	66M	1.94×	-	82.8/82.2	86.8/81.7	88.9	70.4/88.9	65.3	91.8	-
PKD [2019a]	66M	1.94×	43.5	81.5/81.0	85.0/79.9	89.0	70.7/88.9	65.5	92.0	83.4/81.6
BERT-of-Theseus [2020]	66M	1.94×	47.8	82.4/82.1	87.6/83.2	89.6	**71.6/89.3**	66.2	92.2	85.6/84.1
ProKT [2021]	66M	1.94×	-	82.9/82.2	87.0/82.3	89.7	70.9/88.9	-	93.3	-
TinyBERT‡ [2020]	66M	1.94×	47.5	83.0/82.6	87.9/82.8	89.8	70.9/88.6	66.8	93.1	85.8/84.6
DML† [2018b]	66M	1.94×	48.5	82.6/81.6	86.5/81.2	89.5	70.7/88.7	66.3	92.7	85.5/84.0
RCO† [2019]	66M	1.94×	48.2	82.3/81.2	86.8/81.4	89.3	70.4/88.7	66.5	92.6	85.3/84.1
TAKD† [2020]	66M	1.94×	48.4	82.4/81.7	86.5/81.3	89.4	70.6/88.8	66.8	92.9	85.4/84.1
SFTN† [2021]	66M	1.94×	48.1	82.1/81.3	86.5/81.2	89.6	70.2/88.4	66.3	92.7	85.1/84.2
MetaDistil *(ours)*	66M	1.94×	**50.7**	**83.8/83.2**	**88.7/84.7**	**90.2**	71.1/88.9	**67.2**	**93.5**	**86.1/85.0**
w/o pilot update	66M	1.94×	49.1	83.3/82.8	88.2/84.1	89.9	71.0/88.7	66.6	93.5	85.9/84.6

we report Pearson and Spearman correlation. The metric for CoLA is Matthew's correlation. The other tasks use accuracy as the metric.

Following previous works [Sun et al., 2019a, Turc et al., 2019, Xu et al., 2020], we evaluate MetaDistil in a *task-specific* setting where the teacher model is fine-tuned on a downstream task and the student model is trained on the task with the KD loss. We do not choose the pretraining distillation setting since it requires significant computational resources. We implement MetaDistil based on Hugging Face Transformers [Wolf et al., 2020].

Baselines

For comparison, we report the results of vanilla KD and patient knowledge distillation [Sun et al., 2019a]. We also include the results of progressive module replacing [Xu et al., 2020], a state-of-the-art task-specific compression method for BERT which also uses a larger teacher model to improve smaller ones like knowledge distillation. In addition, according to Turc et al. [2019], the reported performance of current task-specific BERT compression methods is underestimated because the student model is not appropriately initialized. To ensure fair comparison, we re-run task-specific baselines with student models initialized by a pretrained 6-layer BERT model and report our results in addition to the official numbers in the original papers. We also compare against deep mutual learning (DML) [Zhang et al., 2018b], teacher assistant knowledge distillation (TAKD) [Mirzadeh et al., 2020], route constraint optimization (RCO) [Jin et al., 2019], proximal knowledge teaching (ProKT) [Shi et al., 2021], and student-friendly teacher network (SFTN) [Park et al., 2021], where the teacher network is not fixed. For reference, we also present results of pretraining distilled models including DistilBERT [Sanh et al., 2019], TinyBERT [Jiao et al., 2020], MiniLM v1 and v2 [Wang et al., 2020, 2021a]. Note that among these baselines, PKD [Sun et al., 2019a] and Theseus [Xu et al., 2020] exploit intermediate features while TinyBERT and the MiniLM family use both intermediate and Transformer-specific features. In contrast, MetaDistil uses none of these but the vanilla KD loss (Equation 5.1).

Training Details

For training hyperparameters, we fix the maximum sequence length to 128 and the temperature to 2 for all tasks. For our method and all baselines (except those with officially reported numbers), we perform grid search over the sets of the student learning rate λ from {1e-5, 2e-5, 3e-5}, the teacher learning rate μ from {2e-6, 5e-6, 1e-5}, the batch size from {32, 64}, the weight of KD loss from {0.4, 0.5, 0.6}. We randomly split the original training set to a new training set and the quiz set by 9 : 1. For RCO, we select four unconverged teacher checkpoints as the intermediate training targets. For TAKD, we use KD to train a teacher assistant model with 10 Transformer layers.

5.3.2 Experimental Results

We report the experimental results on both the development set and test set of the eight GLUE tasks [Wang et al., 2019b] in Table 5.1. MetaDistil achieves state-of-the-art performance under the task-specific setting and outperforms all KD baselines. Notably, without using any intermediate or model-specific features in the loss function, MetaDistil outperforms methods with carefully designed features, e.g., PKD and TinyBERT (without data augmentation). Compared with other methods with a trainable teacher [Zhang et al., 2018b, Mirzadeh et al., 2020, Jin et al., 2019, Shi et al., 2021], our method still demonstrates superior performance. As we analyze, with the help of meta learning, MetaDistil is able to directly optimize the teacher's teaching ability thus yielding a further improvement in terms of student accuracy. Also, we observe a performance drop by replacing pilot update with a normal update. This ablation study verifies the effectiveness of our proposed pilot update mechanism. Moreover, MetaDistil achieves very competitive results on image classification as well, as described in Section 5.6.2.

5.4 Analysis

5.4.1 Why Does MetaDistil Work?

We investigate the effect of meta-update for each iteration. We inspect (1) the validation loss of S' after the teaching experiment and that of S after the real distillation update, and (2) the KD loss, which describes the discrepancy between student and teacher, before and after the teacher update.

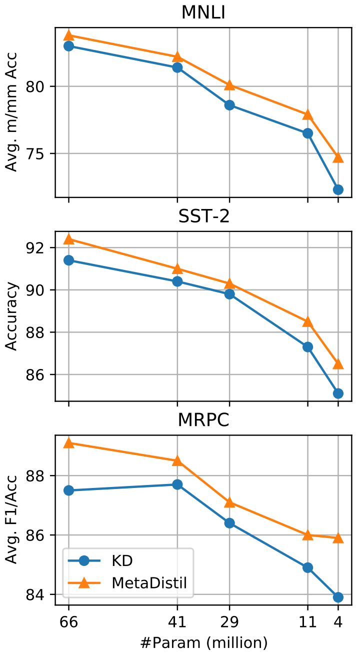


Figure 5.2. Results with different student architectures.

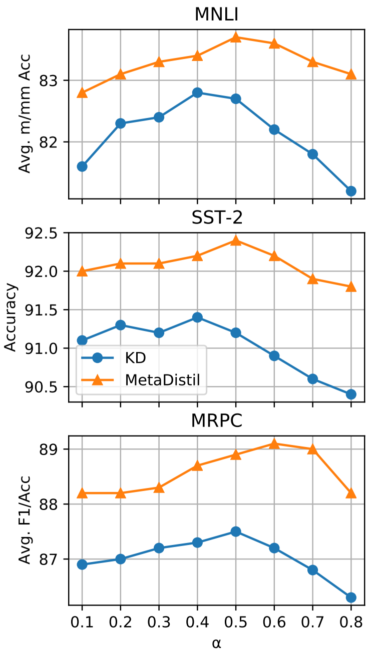


Figure 5.3. Results with different loss weight α.

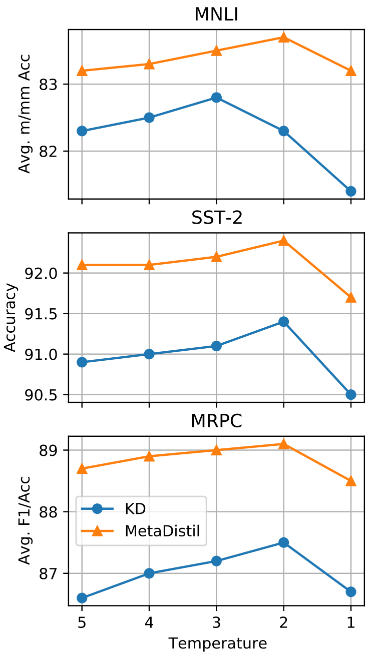


Figure 5.4. Results with different temperature.

We find that for 87% of updates, the student model's validation loss after real update (Line 7 in Algorithm 1) is smaller than that after the teaching experiment (Line 4 in Algorithm 1), which would be the update to the student S in the variant without pilot update. This confirms the effectiveness of the pilot update mechanism on better matching the student and teacher model.

Moreover, we find that in 91% of the first half of the updates, the teacher becomes more similar (in terms of logits distributions) to the student after the meta-update, which indicates that the teacher is learning to adapt to a low-performance student (like an elementary school teacher). However, in the second half of MetaDistil, this percentage drops to 63%. We suspect this is because in the later training stages, the teacher needs to actively evolve itself beyond the student to guide the student towards further improvement (like a university professor).

Finally, we try to apply a meta-learned teacher to a conventional static distillation and also to an unfamiliar student. We describe the results in details in Section 5.6.3.

5.4.2 Hyper-parameter Sensitivity

A motivation of MetaDistil is to enable the teacher to dynamically adjust its knowledge transfer in an optimal way. Similar to Adam [Kingma and Ba, 2015] vs. SGD [Sinha and Griscik, 1971, Kiefer et al., 1952] for optimization, with the ability of dynamic adjusting, it is natural to expect MetaDistil to be more insensitive and robust to changes of the settings. Here, we evaluate the performance of MetaDistil with students of various capability, and a wide variety of hyperparameters, including *loss weight* and *temperature*.

Student Capability To investigate the performance of MetaDistil under different student capacity, we experiment to distill BERT-Base into BERT-6L, Medium, Small, Mini and Tiny [Turc et al., 2019] with conventional KD and MetaDistil. We plot the performance with the student's parameter number in Figure 5.2. Additionally, we show results for different compression ratio in Section 5.7.

Loss Weight In KD, tuning the loss weight is non-trivial and often requires hyperparameter search. To test the robustness of MetaDistil under different loss weights, we run experiments with different α (Equation 5.1). As shown in Figure 5.3, MetaDistil consistently outperforms conventional KD and is less sensitive to different α.

Temperature Temperature is a re-scaling trick introduced in Hinton et al. [2015]. We try different temperatures and illustrate the performance of KD and MetaDistil in Figure 5.4. MetaDistil shows better performance and robustness compared to KD.

5.4.3 Limitation

Like all meta learning algorithms, MetaDistil inevitably requires two rounds of updates involving both first and second order derivatives. Thus, MetaDistil requires additional computational time and memory than a normal KD method, which can be a limitation of our method. We compare the computational overheads of MetaDistil with other methods in Table 5.2. Although our approach takes more time to achieve its own peak performance, it can match up the perfor-

Table 5.2. Comparison of training time and memory cost of MetaDistil with the baselines. "Training Time (Best)" denotes the training time for each method to achieve its own best performance on the development set. "Training Time (Match)" denotes the training time for each method to match the best performance of PKD on the development set. The batch size is 4. All experiments are conducted on a single Nvidia V100 GPU.

Method	PKD [Sun et al., 2019a]	ProKT [Shi et al., 2021]	MetaDistil *(ours)*
Training Time (Best)	13 min.	25 min.	31 min.
Training Time (Match)	13 min.	18 min.	16 min.
Memory Cost	4.2 GB	6.8 GB	11.4 GB
Best Acc/F1	89.4/84.7	90.7/86.3	91.1/86.8

mance of PKD [Sun et al., 2019a] with a similar time cost. The memory use of our method is higher than PKD and ProKT [Shi et al., 2021]. However, this one-off investment can lead to a better student model for inference, thus can be worthy.

5.5 Discussion

In this chapter, we present MetaDistil, a knowledge distillation algorithm powered by meta learning that explicitly optimizes the teacher network to better transfer its knowledge to the student network. The extensive experiments verify the effectiveness and robustness of MetaDistil.

5.6 Additional Results: MetaDistil for Image Classification

In addition to BERT compression, we also provide results on image classification. Also, we conduct experiments of static teaching and cross teaching, to further verify the effectiveness of MetaDistil of adapting to different students.

5.6.1 Experimental Settings

For CV, following the settings in Tian et al. [2020], we experiment with the image classification task on CIFAR-100 [Krizhevsky et al., 2009] with student-teacher combinations of different capacity and architectures, including ResNet [He et al., 2016] and VGG [Simonyan and

Zisserman, 2015]. Additionally, we run a distillation experiment between different architectures (a ResNet teacher to a VGG student). We report the top-1 test accuracy of the compressed student networks. We inherit all hyperparameters from Tian et al. [2020] except for the teacher learning rate, which is grid searched from {1e-4, 2e-4, 3e-4}. We randomly split the original training set to a new training set and the quiz set by 9 : 1. We use the KL loss in Hinton et al. [2015] as the distillation objective. We compare our results with a state-of-the-art distillation method, CRD [Tian et al., 2020] and other commonly used knowledge distillation methods [Hinton et al., 2015, Romero et al., 2015, Zagoruyko and Komodakis, 2017, Tung and Mori, 2019, Peng et al., 2019, Ahn et al., 2019, Park et al., 2019, Passalis and Tefas, 2018, Heo et al., 2019, Kim et al., 2018] including ProKT [Shi et al., 2021] which has a trainable teacher.

5.6.2 Image Recognition Results

We show the experimental results of MetaDistil distilling ResNet [He et al., 2016] and VGG [Simonyan and Zisserman, 2015] with five different teacher-student pairs. MetaDistil achieves comparable performance to CRD [Tian et al., 2020], the current state-of-the-art distillation method on image classification while outperforming all other baselines with complex features and loss functions. Notably, CRD introduces additional negative sampling and contrastive training while our method achieves comparable performance without using these tricks. Additionally, we observe a substantial performance drop without pilot update, again verifying the importance of this mechanism.

5.6.3 Static Teaching and Cross Teaching

In MetaDistil, the student is trained in a dynamic manner. To investigate the effect of such a dynamic distillation process, we attempt to use the teacher at the end of MetaDistil training to perform a static conventional KD, to verify the effectiveness of our dynamic distillation strategy. As shown in Table 5.4, on both experiments, dynamic MetaDistil outperforms conventional KD and static distillation with the teacher at the end of MetaDistil training.

Table 5.3. Experimental results on the test set of CIFAR-100. The best and second best results are marked with **boldface** and underline, respectively. All baseline results except ProKT are reported in Tian et al. [2020]. *ResNet for ImageNet. Other ResNets are ResNet for CIFAR [He et al., 2016].

Teacher **Student**	ResNet-56 ResNet-20	ResNet-110 ResNet-20	ResNet-110 ResNet-32	VGG-13 VGG-8	ResNet-50* VGG-8
Teacher	72.34	74.31	74.31	74.64	79.34
Student	69.06	69.06	71.14	70.36	70.36
KD [Hinton et al., 2015]	70.66	70.67	73.08	72.98	73.81
FitNet [Romero et al., 2015]	69.21	68.99	71.06	71.02	70.69
AT [Zagoruyko and Komodakis, 2017]	70.55	70.22	72.31	71.43	71.84
SP [Tung and Mori, 2019]	69.67	70.04	72.69	72.68	73.34
CC [Peng et al., 2019]	69.63	69.48	71.48	70.71	70.25
VID [Ahn et al., 2019]	70.38	70.16	72.61	71.23	70.30
RKD [Park et al., 2019]	69.61	69.25	71.82	71.48	71.50
PKT [Passalis and Tefas, 2018]	70.34	70.25	72.61	72.88	73.01
AB [Heo et al., 2019]	69.47	69.53	70.98	70.94	70.65
FT [Kim et al., 2018]	69.84	70.22	72.37	70.58	70.29
ProKT [Shi et al., 2021]	70.98	70.74	72.95	73.03	73.90
CRD [Tian et al., 2020]	71.16	**71.46**	**73.48**	**73.94**	74.30
MetaDistil	**71.25**	71.40	73.35	73.65	**74.42**
w/o pilot update	71.02	70.96	73.31	73.48	74.05

As mentioned in Section 5.2.2, a meta teacher is optimized to transfer its knowledge to a specific student network. To justify this motivation, we conduct experiments using a teacher optimized for the ResNet-32 student to statically distill to the ResNet-20 student, and also in reverse. As shown in Table 5.4, the cross-taught students underperform the static students taught by their own teachers by 0.27 and 0.12 for ResNet-32 and ResNet-20, respectively. This confirms our motivation that the meta teacher in MetaDistil can adjust itself according to its student.

5.7 Additional Results: Different Compression Ratios

In this section, we present additional experimental results in settings with different compression ratios to further demonstrate the effectiveness of MetaDistil on bridging the gap between the student and teacher capacity. Specifically, we conduct experiments in the following two settings: (1) distilling BERT-base into a 4-layer BERT (110M→52M) and (2) distilling BERT-large into a 6-layer BERT (345M→66M). The results are shown in Table 4 and Table

Table 5.4. Experimental results of static teaching and cross teaching.

Teacher	**Student**	**Acc@1**
KD (ResNet-110)	ResNet-32 (static)	73.08
	ResNet-20 (static)	70.67
MetaDistil (ResNet-110→ResNet-32)	ResNet-32 (dynamic)	73.35
	ResNet-32 (static)	73.16
	ResNet-20 (static, cross)	70.82
MetaDistil (ResNet-110→ResNet-20)	ResNet-20 (dynamic)	71.40
	ResNet-20 (static)	70.94
	ResNet-32 (static, cross)	72.89

Table 5.5. Experimental results on the development set of GLUE in the setting of distilling BERT-base in to BERT_4. †Results reported by us. All results reported by us are average performance of 3 runs with different random seeds.

Method	**#Param.**	**Speed-up**	**CoLA** (8.5K)	**MNLI** (393K)	**MRPC** (3.7K)	**QNLI** (105K)	**QQP** (364K)	**RTE** (2.5K)	**SST-2** (67K)	**STS-B** (5.7K)
BERT-Base (teacher) [2019]	110M	1.00×	58.9	84.6/84.9	91.6/87.6	91.2	88.5/91.4	71.4	93.0	90.2/89.8
BERT_4-KD† [2015]	55M	2.90×	32.5	80.5/80.9	87.2/83.1	87.5	86.6/90.4	65.2	90.2	84.5/84.2
BERT_4-PKD† [2019a]	55M	2.90×	34.2	80.9/81.3	87.0/82.9	87.7	86.8/90.5	66.1	90.5	84.3/84.0
BERT_4-ProKT† [2021]	55M	2.90×	36.6	81.4/81.9	87.6/83.5	88.0	87.1/90.5	66.8	90.7	85.2/85.1
MetaDistil $_4$ *(ours)*	55M	2.90×	**40.3**	**82.4/82.7**	**88.4/84.2**	**88.6**	**87.8/90.8**	**67.8**	**91.8**	**86.3/86.0**

5, respectively. We can see that MetaDistil consistently outperforms PKD and ProKT in both settings. This confirms the effectiveness of MetaDistil and also show its ability to adapt the teacher model to the student model, since the gap between teacher and student is even larger in these settings.

Table 5.6. Experimental results on the development set of GLUE in the setting of distilling BERT-large in to BERT_6. †Results reported by us. All results reported by us are average performance of 3 runs with different random seeds.

Method	**#Param.**	**Speed-up**	**CoLA** (8.5K)	**MNLI** (393K)	**MRPC** (3.7K)	**QNLI** (105K)	**QQP** (364K)	**RTE** (2.5K)	**SST-2** (67K)	**STS-B** (5.7K)
BERT-Large (teacher) [2019]	345M	1.00×	71.5	86.5/86.7	92.5/88.7	92.5	89.6/91.8	73.4	94.5	91.2/90.6
BERT_6-KD† [2015]	66M	3.88×	58.8	82.8/83.0	89.6/85.0	89.5	87.5/91.0	68.0	91.1	88.5/88.4
BERT_6-PKD† [2019a]	66M	3.88×	59.2	82.9/83.1	89.9/85.4	89.8	87.9/91.1	67.9	91.5	88.2/88.0
BERT_6-ProKT† [2021]	66M	3.88×	59.8	83.2/83.4	91.0/86.5	90.0	88.2/91.0	68.8	91.6	88.7/88.5
MetaDistil $_6$ *(ours)*	66M	3.88×	**63.5**	**83.9/84.3**	**91.5/87.3**	**90.8**	**88.7/91.3**	**70.8**	**92.9**	**89.6/89.4**

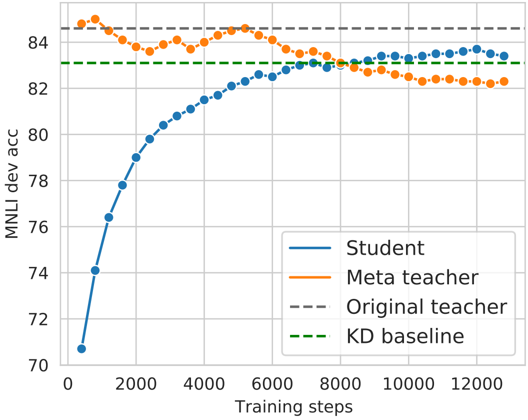


Figure 5.5. Learning dynamics of the student and teacher in MetaDistil on the development set of MNLI.

5.8 Additional Results: Distillation Dynamics

We also investigate why MetaDistil works by conducting experiments on the development sets of MNLI, SST, and MRPC, which are important tasks in GLUE that have a large, medium, and small training set, respectively.

We illustrate the validation accuracy curves of the meta teacher and student models with training steps in Figure 5.5, and compare them to the student performance in conventional KD. We can see that the meta teacher maintains high accuracy in the first 5,000 steps and then begins to slowly degrade. Starting from step 8,000, the teacher model underperforms the student while the student's accuracy keeps increasing. This verifies our assumption that a model with the best accuracy is not necessarily the optimal teacher. Also, MetaDistil is not naively optimizing the teacher's accuracy but its "teaching skills." This phenomenon suggests that beyond high accuracy, there could be more important properties of a good teacher that warrant further investigation.

5.9 Additional Results: Improvement Analysis

While MetaDistil achieves improved student accuracy on the GLUE benchmark, it is still not very clear where the performance improvement comes from. There are two possibilities:

(1) the student better mimics the teacher, and (2) the changes of teacher helps student perform better on hard examples that would be incorrectly classified by the student with vanilla KD. We conduct a series of analysis on the MRPC dataset.

For the first assumption, we compute the prediction loyalty [Xu et al., 2021a] of the student model distilled with PKD and MetaDistil, respectively. For MetaDistil, we measure the loyalty with respect to both the original teacher and the final teacher. We find that there is no significant difference between between PKD and MetaDistil. This suggests that the improvement does not come from student better mimicking the teacher.

For the second assumption, we first identify the examples in the quiz set for which our model gives correct predictions while the student distilled by PKD makes a wrong prediction. We then compute the loss (cross entropy) of the original teacher and the teacher updated by MetaDistil. We find the loss is substantially reduced by MetaDistil. In contrast, the overall loss of teacher on the development set does not decrease. This suggests that MetaDistil can help the teacher concentrate on hard examples that the student struggles in the quiz set and learn to perform better on these examples, thus facilitate student learning.

This chapter, in part, is a reprint of the material as it appears in “BERT Learns to Teach: Knowledge Distillation with Meta Learning” by Wangchunshu Zhou, Canwen Xu, and Julian McAuley, which was published at *Association for Computational Linguistics*, 2022. The dissertation author was the co-primary investigator and co-first author of this paper.

Chapter 6

Automatic Multi-Label Prompting: Simple and Interpretable Few-Shot Classification

Prompt-based learning (i.e., prompting) is an emerging paradigm for exploiting knowledge learned by a pretrained language model. In this chapter, we propose Automatic Multi-Label Prompting (AMuLaP), a simple yet effective method to automatically select label mappings for few-shot text classification with prompting. Our method exploits one-to-many label mappings and a statistics-based algorithm to select label mappings given a prompt template. Our experiments demonstrate that AMuLaP achieves competitive performance on the GLUE benchmark without human effort or external resources.[1]

6.1 Introduction

Since the release of GPT-3 [Brown et al., 2020], several studies have focused on exploiting pretrained language models with only a few training examples [Brown et al., 2020, Gao et al., 2021b, Shin et al., 2020]. These works demonstrate the potential of using natural language prompts to encourage the model to recall similar patterns in its training corpus and thus make accurate predictions. This setting of few-shot learning is closer to how humans learn to solve a task, often without many examples as in a traditional deep learning paradigm. The use of prompts can strengthen the explicit connection between input and output, helping the model exploit the

---

[1] The code is available at https://github.com/HanNight/AMuLaP.

knowledge learned from pretraining in a better way. Furthermore, recent works [Schick and Schütze, 2021a,b, Gao et al., 2021b] show that prompts can also help the model generalize better in fine-tuning.

Prompt-based learning (i.e., prompting) aims to use a template to convert the original input into a prompt-based input with some unfilled masked tokens, and then use the pretrained language model to fill these masked tokens, and finally the tokens filled into these slots are mapped to the corresponding labels as the final output. In prompting, the design of prompts often plays an important role. Many attempts have been made in this emerging direction of *prompt engineering* [Shin et al., 2020, Gao et al., 2021b]. Meanwhile, finding a good mapping from the original task labels to tokens (i.e., *label engineering*) is also critical to few-shot performance, as found in Schick et al. [2020], Gao et al. [2021b]. However, manually assigning the label mapping requires human expertise with trial and error. One may argue that the same effort can be used to label more supervised data for a conventional deep learning pipeline. Thus, an efficient automatic label mapping method is desirable.

In this paper, we aim to design a method that can automatically find a good label mapping to save human effort from label engineering. We propose **A**utomatic **Mu**lti-**La**bel **P**rompting (AMuLaP), a simple yet effective method to tackle the label selection problem for few-shot classification. AMuLaP is a parameter-free statistical technique that can identify the label patterns from a few-shot training set given a prompt template. AMuLaP exploits multiple labels to suppress the noise and inherently extend the training set for prompt-based fine-tuning. Compared with a hand-crafted label mapping and previous works on automatic label mapping [Schick et al., 2020, Gao et al., 2021b], AMuLaP achieves competitive performance despite being simpler and does not require access to the weights of the backbone model, or finetune an external pretrained language model for searching label mapping. We conduct extensive experiments and demonstrate the effectiveness of our method under multiple settings. Moreover, we attempt to scale AMuLaP with different sizes of the training set and find AMuLaP to work surprisingly well even with one or two shots. We further analyze why does AMuLaP work and discuss the pros and cons of

Table 6.1. The manual and automatically selected labels by AMuLaP. The templates used for prompting are from Gao et al. [2021b].

Task	Template	Class	Manual [Gao et al., 2021b]	Labels selected by AMuLaP

MNLI	<S_1> ? `[MASK]` , <S_2>	`entailment`
		`neutral`
		`contradiction`
			Yes
			Maybe
			No
				Yes, Indeed, Also, Currently
				Historically, Suddenly, Apparently, And
				No, However, Instead, Unfortunately

SST-2	<S_1> It was `[MASK]` .	`positive` `negative`	great terrible	great, perfect, fun, brilliant terrible, awful, disappointing, not
QNLI	<S_1> ? `[MASK]` , <S_2>	`entailment` `not_entailment`	Yes No	Yes, Historically, Overall, Indeed Well, First, However, Unfortunately
RTE	<S_1> ? `[MASK]` , <S_2>	`entailment` `not_entailment`	Yes No	Yes, Today, Specifically, Additionally However, Ironically, Also, Indeed
MRPC	<S_1> `[MASK]` , <S_2>	`equivalent` `not_equivalent`	Yes No	`</s>`, Currently, Additionally, Today However, Meanwhile, Overall, Finally
QQP	<S_1> `[MASK]` , <S_2>	`equivalent` `not_equivalent`	Yes No	Or, So, Specifically, Actually Also, And, Finally, Well
CoLA	<S_1> This is `[MASK]` .	`grammatical` `not_grammatical`	correct incorrect	why, true, her, amazing it, ridiculous, interesting, sad

prompting as a new paradigm.

6.2 Prompting for Few-Shot Classification

We follow the setup in LM-BFF [Gao et al., 2021b] for few-shot text classification. Given a pretrained language model $\mathscr{L}$, a task $\mathscr{D}$ and its defined label space $\mathscr{Y}$, we have n training examples per class for the training set $\mathscr{D}_{train}$. As pointed out in Perez et al. [2021], using the *full* development set may be misleading to claim a few-shot setting. Thus, we use a *few-shot* development set with the same size as the training set (i.e., $|\mathscr{D}_{train}| = |\mathscr{D}_{dev}|$), to be consistent with Gao et al. [2021b] and constitute a "true few-shot" setting [Perez et al., 2021].

For an input example x (a single sentence or a sentence pair), we first use a task-specific template $\mathscr{T}$ to convert it to x', a token sequence with a `[MASK]` token. We then map the original label space to a set of selected words from the vocabulary, denoted as $\mathscr{M} : \mathscr{Y} \to \mathscr{V}'$. Some examples of $\mathscr{T}$ and $\mathscr{M}$ are shown in Table 6.1. Note that since we focus on automatically finding the label mapping $\mathscr{M}$, we use the manual templates $\mathscr{T}$ from Gao et al. [2021b] throughout this paper. Since $\mathscr{L}$ is trained to complete the `[MASK]` token in an input sequence, we can directly

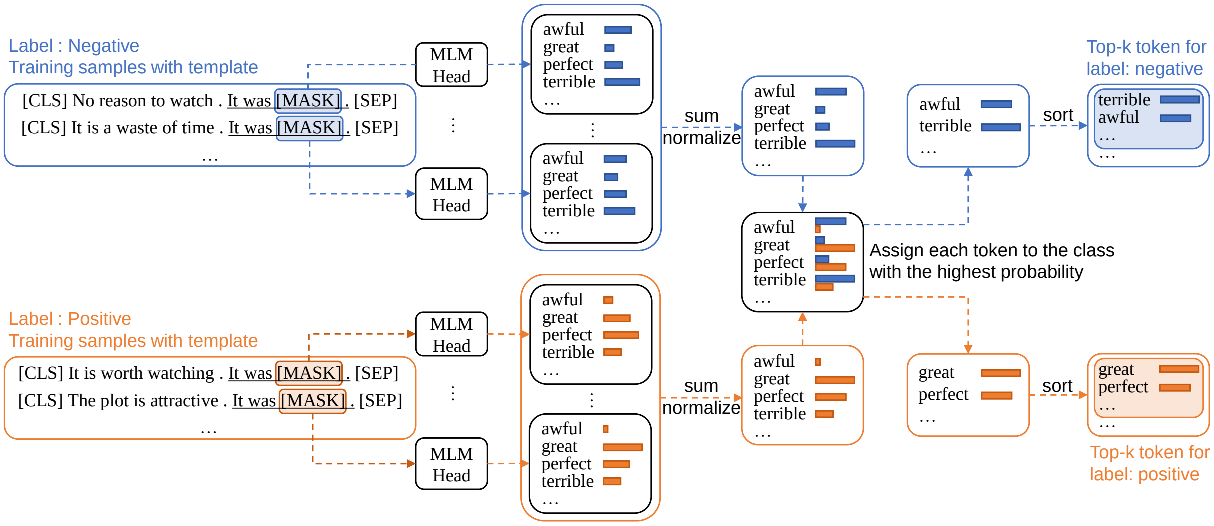

Label : Negative
Training samples with template
[CLS] No reason to watch . It was [MASK] . [SEP]
[CLS] It is a waste of time . It was [MASK] . [SEP]
…
MLM Head
MLM Head
awful
great
perfect
terrible
…
sum
normalize
Top-k token for label: negative
sort
terrible
awful
Assign each token to the class with the highest probability
Label : Positive
Training samples with template
[CLS] It is worth watching . It was [MASK] . [SEP]
[CLS] The plot is attractive . It was [MASK] . [SEP]
…
great
perfect
Top-k token for label: positive

Figure 6.1. The illustration of implementing AMuLaP on a binary sentiment classification task (SST-2). Each training sample with the task-specific template (the underlined text) is fed into a pretrained language model $\mathscr{L}$ to get its own probability distribution over the vocabulary $\mathscr{V}$. All the obtained probability distributions are summed by class and normalized to get the probability distribution of each class. Then each token in $\mathscr{V}$ is assigned to the class with the highest probability (e.g., the token *terrible* is assigned to the class *negative*, the token *great* is assigned to the class *positive*). Finally, for each class, we choose the top-k tokens as label words.

make zero-shot prediction of the probability of class $y \in \mathscr{Y}$ by the masked language modeling:

$$p(y|x) = p\left(\texttt{[MASK]} = \mathscr{M}(y) \mid x'\right). \tag{6.1}$$

Alternately, one can further fine-tune $\mathscr{L}$ with supervised pairs $\{x', \mathscr{M}(y)\}$ to achieve even better performance.

6.3 Automatic Multi-Label Prompting

6.3.1 Exploiting Multiple Labels

Selecting one label word can be insufficient for some complicated tasks, as mentioned in Schick et al. [2020]. We also argue that selecting only one label (especially automatically) may bring noise. This can be resolved by introducing multiple label words. Schick et al. [2020] use multiple label combinations for PET [Schick and Schütze, 2021a] and ensemble them afterwards.

We instead use a straightforward sum to consider multiple label words when making predictions. This design has a similar advantage of exploiting multiple labels without training and ensembling multiple models.

Instead of a one-to-one mapping from the original label space $\mathscr{Y}$ to $\mathscr{V}$, we map each $y \in \mathscr{Y}$ to its label word set $\mathscr{S}(y)$ of k words. We denote the mapping function as $\mathscr{M}' : \mathscr{Y} \to \mathscr{V}^k$. For class $y \in \mathscr{Y}$, the predicted probability is calculated as:

$$p(y|x) = \sum_{v \in \mathscr{S}(y)} p\left(\texttt{[MASK]} = v \mid x'\right) \tag{6.2}$$

Then, we can simply make predictions by selecting the label with the largest likelihood.

Similarly, if we need to fine-tune $\mathscr{L}$ with supervised pairs, instead of optimizing the cross-entropy loss between the gold label and a single token, we optimize the loss between the sum of the output probabilities of $\mathscr{S}(y)$ and the gold label with a cross-entropy loss:

$$l = -\sum_{x \in \mathscr{D}_{train}} \sum_{y \in \mathscr{Y}} \left[\mathbb{1}\left[y = \hat{y}\right] \cdot \log p(y|x)\right] \tag{6.3}$$

where $\hat{y}$ is the ground truth label for the input x and $p(y|x)$ is defined in Equation 6.2.

6.3.2 Automatic Label Selection

Finding a good label mapping $\mathscr{M}$ is non-trivial, especially when $\mathscr{M}'$ maps an original label to a set of label words instead of one. Selecting a good label mapping often requires significant human effort, including domain knowledge and trial-and-error. Previously, Schick and Schütze [2021a,b] both use hand-crafted label mappings while Schick et al. [2020] explores automatic label mapping searching but it still requires manual pre-filtering and significantly underperforms the manual mapping. [Gao et al., 2021b] exploits a large pretrained masked language model (RoBERTa, Liu et al., 2019b) to construct a pruned set of label words and then determine the final mapping by fine-tuning on all of them and selecting the best one with $\mathscr{D}_{dev}$.

We introduce a new selection algorithm for label mapping that achieves competitive results compared to previous efforts.

We aim to achieve two goals: **(1) Selecting the most likely label mapping based on the training set.** For example, in a sentiment classification task, we would like to see positive words in the label set of the "positive" class while negative words in the label set of the "negative" class. A simple solution is to select the k most likely tokens predicted for the `[MASK]` token in the training examples of each class y. However, in practice, we would find common words in more than one label set. For example, if we simply take the 10 most likely tokens for the SST-2 dataset [Socher et al., 2013], we would find "good" in both positive and negative label sets, although it is ranked second place in the positive set and ninth in the negative set. Thus, we want to make sure that **(2) Each token only belongs to at most one label set where it has the highest probability.** To ensure this, we have to iterate over the vocabulary and check that for every token. Then, we can truncate the candidate sets of each class and select the k most likely tokens from each set. The time complexity of this algorithm is $O(k \cdot |\mathscr{V}| \cdot |\mathscr{Y}|)$.

Formally, we select $\mathscr{M}' : \mathscr{Y} \rightarrow \mathscr{V}^k$ by the following steps:

1. For each $y_i \in \mathscr{Y}$, we iterate through all training samples $x_j \in \mathscr{D}_{train}$ whose ground truth label $\hat{y}_j = y_i$. We use $\mathscr{L}$ to predict the token probability of the `[MASK]` token and take the average of the predicted probabilities of the n examples to be $\mathbf{z}_i$, where $\mathbf{z}_i$ is a vector over the whole vocabulary.
2. For each $y_i \in \mathscr{Y}$, initialize an empty candidate token set $\tilde{\mathscr{S}}(y_i)$.
3. For each $v \in \mathscr{V}$ where $\mathscr{V}$ is the vocabulary of the model $\mathscr{L}$, we retrieve v's probability value z_i^v from $\mathbf{z}_i$ of each class.
4. We assign v to the most likely token set of the m-th class $\tilde{\mathscr{S}}(y_m)$ where $m = \arg\max_i z_i^v$.
5. For $y_i \in \mathscr{Y}$, we choose the top-k tokens from $\tilde{\mathscr{S}}(y_i)$ with the largest probability z_i^v and obtain the truncated word set $\mathscr{S}(y_i)$.

Table 6.2. Experimental results under three settings with RoBERTa-large as $\mathscr{L}$. For few-shot settings, n is set to 16 per class. We report the average of 5 runs along with their standard deviation in the parentheses.

	MNLI (acc)	**MNLI-mm** (acc)	**SST-2** (acc)	**QNLI** (acc)	**RTE** (acc)	**MRPC** (F1)	**QQP** (F1)	**CoLA** (Matt.)	**Avg.**
Baselines									
Majority	32.7	33.0	50.9	49.5	52.7	81.2	0.0	0.0	37.5
Manual Label 0-shot [2021b]	50.8	51.7	83.6	50.8	51.3	61.9	49.7	2.0	50.2
Full Fine-tuning	89.8	89.5	95.0	93.3	80.9	91.4	81.7	62.6	85.5
Setting 1: *$\mathscr{D}_{train}$ only; No parameter update.*									
In-context learning [2020]	**52.0** (0.7)	**53.4** (0.6)	84.8 (1.3)	**53.8** (0.4)	**60.4** (1.4)	45.7 (6.0)	36.1 (5.2)	-1.5 (2.4)	48.1 (2.3)
AMuLaP (ours)	50.8 (2.1)	52.3 (1.8)	**86.9** (1.6)	53.1 (2.8)	58.9 (7.9)	**56.3** (5.0)	**60.2** (2.7)	**2.3** (1.4)	**52.6** (3.2)
Setting 2: *$\mathscr{D}_{train}$ + $\mathscr{D}_{dev}$; No parameter update.*									
PETAL-CE [2020]	48.8 (2.6)	49.7 (2.3)	75.6 (7.2)	49.5 (0.0)	**63.5** (3.3)	28.9 (39.6)	59.2 (0.0)	1.3 (3.0)	47.1 (7.3)
PETAL-LR [2020]	38.6 (2.0)	38.4 (2.1)	85.3 (3.3)	53.3 (3.6)	54.7 (6.4)	28.0 (38.5)	55.6 (2.8)	1.5 (3.4)	44.4 (7.8)
Auto-L [2021b]	41.6 (5.4)	42.3 (6.2)	84.3 (3.3)	57.9 (3.9)	61.9 (7.5)	**67.7** (7.9)	55.5 (5.0)	1.2 (4.8)	51.6 (5.5)
AMuLaP (ours)	50.8 (2.1)	52.2 (1.9)	87.0 (1.5)	53.5 (2.3)	59.1 (7.4)	56.7 (5.7)	**61.5** (1.7)	2.6 (1.8)	52.9 (3.1)
Auto-L + AMuLaP (ours)	**52.9** (3.0)	**54.2** (2.7)	**90.1** (0.4)	**57.9** (2.6)	59.9 (5.2)	66.0 (3.0)	59.4 (2.3)	**2.7** (5.7)	**55.4** (3.1)
Setting 3: *$\mathscr{D}_{train}$ + $\mathscr{D}_{dev}$; Prompt-based fine-tuning.*									
Fine-tuning	45.8 (6.4)	47.8 (6.8)	81.4 (3.8)	60.2 (6.5)	54.4 (3.9)	76.6 (2.5)	60.7 (4.3)	**33.9** (14.3)	57.6 (6.1)
Manual Label FT [2021b]	68.3 (2.3)	70.5 (1.9)	92.7 (0.9)	64.5 (4.2)	69.1 (3.6)	74.5 (5.3)	65.5 (5.3)	9.3 (7.3)	64.3 (3.9)
PETAL-CE FT [2020]	57.5 (3.2)	57.7 (2.6)	92.6 (1.0)	50.5 (0.0)	68.6 (6.5)	32.1 (42.5)	66.7 (3.2)	3.8 (8.4)	53.7 (8.4)
PETAL-LR FT [2020]	64.0 (6.5)	65.9 (6.4)	92.9 (1.7)	65.5 (6.8)	63.3 (7.7)	77.7 (3.9)	65.7 (4.2)	11.9 (7.5)	63.4 (5.6)
Auto-L FT [2021b]	64.8 (4.7)	67.3 (4.3)	**93.5** (0.5)	**69.8** (3.0)	67.4 (3.9)	76.2 (4.8)	66.4 (4.5)	23.2 (17.1)	66.1 (5.4)
AMuLaP FT (ours)	**70.6** (2.7)	**72.5** (2.4)	93.2 (0.7)	65.1 (5.9)	65.9 (6.3)	**79.3** (4.0)	**69.1** (2.5)	18.3 (9.4)	**66.8** (4.2)
Auto-L + AMuLaP FT (ours)	68.5 (2.2)	71.1 (2.3)	93.4 (1.0)	69.6 (1.1)	**69.4** (4.0)	75.5 (5.6)	66.4 (3.0)	14.2 (14.0)	66.0 (4.2)

The entire workflow is illustrated in Figure 6.1.

6.4 Experiments

6.4.1 Experimental Setting

Datasets

We evaluate seven classification tasks of the GLUE benchmark [Wang et al., 2019b]. Specifically, we test on Microsoft Research Paraphrase Matching (MRPC) [Dolan and Brockett, 2005], Quora Question Pairs (QQP) for Paraphrase Similarity Matching; Stanford Sentiment Treebank (SST-2) [Socher et al., 2013] for Sentiment Classification; Multi-Genre Natural Language Inference Matched (MNLI-m), Multi-Genre Natural Language Inference Mismatched (MNLI-mm) [Williams et al., 2018], Question Natural Language Inference (QNLI) [Rajpurkar et al., 2016] and Recognizing Textual Entailment (RTE) [Wang et al., 2019b] for the Natural Language Inference (NLI) task; The Corpus of Linguistic Acceptability (CoLA) [Warstadt et al., 2019] for Linguistic Acceptability. We use the manual templates in Gao et al. [2021b], as listed in Table 6.1. The metrics for each dataset are indicated in Table 6.2.

Baselines

We compare our method to various baselines:

- **Majority**: always predict the majority class in the test set.
- **GPT-3-style in-context learning** [Brown et al., 2020]: present a few examples to the language model and make it directly predict the next token as the prediction.
- **Manual prompts**: we use the human-designed prompts in Gao et al. [2021b].
- **PETAL-CE** [Schick et al., 2020]: the variant of PETAL using the cross-entropy metric.
- **PETAL-LR** [Schick et al., 2020]: the variant of PETAL using the likelihood ratio metric.
- **Auto-L** [Gao et al., 2021b]: the automatic label searching method with an external pretrained language model, RoBERTa-large [Liu et al., 2019b]. Note that the results of this baseline is different from those reported in Table 3 of Gao et al. [2021b] since they search for both templates and label mapping whereas we fix the templates and search for the label mapping alone, for the sake of fair comparison. We use the officially released code and same hyperparameters for this baseline.

Task Setup

We closely follow the setup in Gao et al. [2021b]. We sample n training examples and n development examples per class. We set $k = 16$ throughout all experiments. We use RoBERTa-large [Liu et al., 2019b] as the backbone LM $\mathscr{L}$. For each reported result, we measure average performance across 5 different randomly sampled $\mathscr{D}_{train}$ and $\mathscr{D}_{dev}$ splits. Following Gao et al. [2021b], the original development split of each dataset is used as the test set in our experiments. We also report the standard deviation for each result. To fairly compare with different baselines, we consider the following three settings:

Table 6.3. Most likely label mapping for the SST-2 dataset obtained by PETAL [Schick et al., 2020], Auto-L [Gao et al., 2021b] and our AMuLaP. Suitable labels annotated by the human annotator are underlined.

Class	**PETAL-CE** [Schick et al., 2020]	**PETAL-LR** [Schick et al., 2020]
`positive`	amazing, great, brilliant, perfect, fun, wonderful, beautiful, fantastic, awesome, not	superb, fearless, acclaimed, addictive, visionary, immersive, irresistible, timely, unforgettable, gripping
`negative`	not, awful, fun, funny, terrible, great, amazing, hilarious, awesome, good	annoying, insulting, meaningless, lame, shitty, humiliating, childish, stupid, embarrassing, irritating
Class	**Auto-L** [Gao et al., 2021b]	**AMuLaP** (ours)
`positive`	exquisite, perfection, effective, fabulous, intense inspiring, spectacular, sublime, astounding, thrilling	great, perfect, fun, brilliant, amazing, good, wonderful, beautiful, excellent, fantastic
`negative`	embarrassing, boring, frustrating, ridiculous, awkward silly, nothing, disgusting, ugly, confusing	terrible, awful, disappointing, not, horrible, obvious, funny, inevitable, bad, boring

- **Setting 1**: We only use $\mathscr{D}_{train}$ alone for both label selection and tuning k. The parameters of $\mathscr{L}$ are not updated. $\mathscr{D}_{dev}$ is not used. This setting is for fair comparison with *In-context learning*.

- **Setting 2**: We use $\mathscr{D}_{train}$ for label selection and an additional $\mathscr{D}_{dev}$ for k tuning. The parameters of $\mathscr{L}$ are not updated. This setting is for fair comparison with Auto-L [Gao et al., 2021b] and PETAL [Schick et al., 2020].

- **Setting 3**: We use $\mathscr{D}_{train}$ and $\mathscr{D}_{dev}$ in the same way as Setting 2 but fine-tune the parameters of the language model $\mathscr{L}$. This setting is for fair comparison with conventional fine-tuning, prompt-based fine-tuning with manual prompts, Auto-L [Gao et al., 2021b] and PETAL [Schick et al., 2020].

Implementation Details

We implement AMuLaP based on Hugging Face Transformers [Wolf et al., 2020]. When selecting k, if there are multiple k with identical performance (which happens occasionally given there are only 16 examples for each class in $\mathscr{D}_{dev}$), we always choose the largest k. For Settings 1 and 2, we search k over $\{1, 2, 4, \ldots, 1024\}$. Note that for settings that do not update the parameters of $\mathscr{L}$, search over k is fast, as we only need to run the model once and cache the distribution of the `[MASK]` token. For prompt-based fine-tuning (Setting 3), where we fine-tune

Table 6.4. Experimental results for the ablation study. We report the average of 5 runs along with their standard deviation in the parentheses.

	MNLI (acc)	**MNLI-mm** (acc)	**SST-2** (acc)	**QNLI** (acc)	**RTE** (acc)	**MRPC** (F1)	**QQP** (F1)	**CoLA** (Matt.)	**Avg.**
Setting 2: *$\mathscr{D}_{train}$ + $\mathscr{D}_{dev}$; No parameter update.*									
AMuLaP	**50.8** (2.1)	**52.2** (1.9)	87.0 (1.5)	53.5 (2.3)	**59.1** (7.4)	**56.7** (5.7)	**61.5** (1.7)	**2.6** (1.8)	**52.9** (3.1)
w/o dedup.	45.4 (2.7)	46.5 (2.5)	**87.9** (1.0)	**53.8** (3.0)	54.6 (6.0)	66.7 (12.3)	57.2 (2.1)	2.5 (4.2)	51.8 (4.2)
$k = 1$	46.5 (2.7)	48.4 (2.6)	68.8 (12.0)	51.9 (1.6)	58.8 (12.7)	55.0 (4.8)	59.2 (0.0)	5.6 (2.1)	49.3 (4.8)
Setting 3: *$\mathscr{D}_{train}$ + $\mathscr{D}_{dev}$; Prompt-based fine-tuning.*									
AMuLaP FT	**70.6** (2.7)	**72.5** (2.4)	**93.2** (0.7)	65.1 (5.9)	**65.9** (6.3)	79.3 (4.0)	**69.1** (2.5)	18.3 (9.4)	**66.8** (4.2)
w/o dedup.	56.9 (5.4)	58.2 (5.2)	92.8 (0.9)	50.6 (0.4)	57.1 (10.8)	79.2 (3.6)	55.0 (26.0)	5.6 (7.1)	56.9 (7.4)
$k = 1$	67.7 (4.1)	69.8 (3.8)	92.6 (1.0)	**65.9** (5.2)	63.1 (8.0)	**80.2** (3.8)	66.7 (3.2)	19.3 (15.5)	65.7 (5.6)
random $\mathscr{M}'$	58.8 (6.2)	61.1 (6.2)	92.1 (2.1)	62.1 (7.1)	57.0 (11.2)	74.7 (9.2)	60.8 (5.8)	**31.0** (13.9)	62.2 (7.7)
random $\mathscr{M}'$ ($k = 1$)	52.6 (7.8)	55.4 (8.3)	89.0 (4.9)	65.2 (4.5)	55.2 (6.2)	73.4 (10.6)	60.7 (3.7)	17.3 (14.7)	58.6 (7.6)

the model $\mathscr{L}$, we search k in a smaller space $\{1,2,4,8,16\}$ due to the increased computational overhead. Following [Gao et al., 2021b], we grid search the learning rate from {1e-5, 2e-5, 5e-5} and batch size from {2, 4, 8}.

6.4.2 Experimental Results

We demonstrate experimental results under three settings in Table 6.2. Under Setting 1, AMuLaP outperforms GPT-3-style in-context learning by 4.5 in terms of the average score and outperforms zero-shot inference with manually designed labels by 2.4. Under Setting 2, compared to variants of PETAL [Schick et al., 2020], AMuLaP has an advantage of 5.8 and 8.5 in terms of the average score over CE and LR, respectively. Notably, AMuLaP even outperforms Auto-L by 1.3 without using any external model or data. Additionally, we attempt to replace the predicted token distribution of AMuLaP with the validation score of all fine-tuned assignments [Gao et al., 2021b].[2] With the help of many trials in automatic search, AMuLaP outperforms Auto-L by a considerable margin of 3.8 in terms of the average score, verifying the versatility of our multi-label mechanism and label selection algorithm. Under Setting 3, AMuLaP FT outperforms all baselines including Auto-L. Generally speaking, methods with parameter update (Setting 3) have better performance than those that do not require access to parameters.

---

[2] The validation scores of all fine-tuned assignments are obtained on $\mathscr{D}_{dev}$, as described in Gao et al. [2021b]. No external data used. All of these we use are from https://github.com/princeton-nlp/LM-BFF/tree/main/auto_label_mapping.

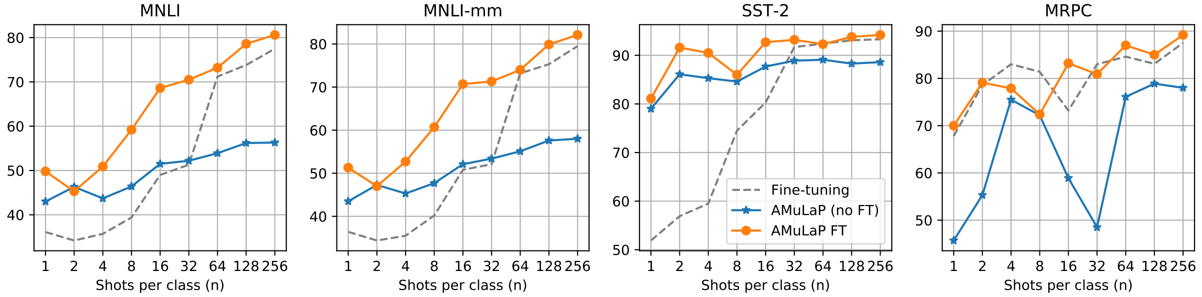


Figure 6.2. Comparison of AMuLaP, AMuLaP FT and fine-tuning on MNLI, SST and MRPC with different n for the training set and the development set.

On all tasks except CoLA, AMuLaP outperforms direct few-shot fine-tuning, suggesting that prompting is a promising method for exploiting large pretrained LMs.

6.5 Analysis

6.5.1 Case Study

As shown in Table 6.3, we list the 10 most likely label mappings output by PETAL [Schick et al., 2020], Auto-L [Gao et al., 2021b] and AMuLaP for the SST-2 dataset, respectively. We shuffle the labels from each model and ask a human annotator to annotate whether they are suitable mappings. PETAL-CE suffers from incorrect mappings for “negative” while PETAL-LR occasionally outputs vague labels. AMuLaP achieves interpretability that is competitive to automatic labels obtained by a fine-tuned pretrained language model, measured by the human agreement ratio. Although AMuLaP outputs three labels that are rated not suitable by the human annotator, it should be noted that all three tokens are ranked low in the candidate set. Thus, introducing top-k truncation can resolve the problem. Additionally, we would like to highlight that AMuLaP mainly collects common words while other methods prefer rare words. This may explain why AMuLaP works well, especially for the non-finetuning settings.

6.5.2 Ablation Study

As shown in Table 6.4, we evaluate the effect of each design choice on the GLUE benchmark. For both non-finetuning and prompt-based fine-tuning settings, our deduplication algorithm can effectively improve the overall performance by 1.1 and 9.9 in terms of the GLUE average score, respectively. Notably, deduplication is especially important for prompt-based fine-tuning since if the same label maps to two classes, optimization would be difficult due to the contradiction of supervision signals. Also, our multi-label strategy is shown to be effective at improving the average GLUE scores by 3.6 and 1.1 for non-finetuning and fine-tuning settings, respectively. Moreover, a random label mapping often leads to lower performance than a label mapping selected based on the training set. An interesting exception is that for CoLA, the random mapping outperforms all label selection methods in Table 6.2 (both manual and automatic) and is close to the fine-tuning baseline.

6.5.3 Scaling Few-Shot Learning

Le Scao and Rush [2021] explore the scaling law of PET [Schick and Schütze, 2021a] when using more examples for training. Similarly, in this section, we aim to test how AMuLaP scales to different training set sizes n. Figure 6.2 illustrates how standard fine-tuning and our AMuLaP with non-finetuning and fine-tuning compare as n increases. For MNLI and SST-2 task, AMuLaP outperforms standard fine-tuning when we use no more than 16 training examples for non-finetuning and fine-tuning setting. When using more than 16 training examples, AMuLaP under fine-tuning setting still outperforms standard fine-tuning. For an easier task like SST-2, although only 32 training examples are used, the performance of our AMuLaP with non-finetuning and fine-tuning is close to saturation and can be comparable to standard fine-tuning on the entire dataset. For a harder task like MNLI, although the performance of AMuLaP under non-finetuning setting gradually becomes saturated as n increases, AMuLaP under fine-tuning settings continues to improve as n increases and continues to outperform the standard fine-tuning.

For MRPC, although the performance of our AMuLaP and standard fine-tuning fluctuate as n increases, in general, AMuLaP with fine-tuning can still achieve comparable performance to standard fine-tuning. In addition, the results demonstrate the effectiveness of AMuLaP especially for extreme few-shot settings. With only one example, AMuLaP achieves decent performance while standard fine-tuning is close to random.

6.6 Discussion

Why Does AMuLaP Work?

Schick et al. [2020] argues that one single label sometimes cannot represent all examples in a class, and thus multiple labels are needed. However, we find this explanation insufficient for understanding the mechanism behind the improved performance with multiple labels. Under a few-shot setting, the limited number of training examples n and complex training procedure of the backbone model $\mathscr{L}$ can often bring noise to both automatic label selection and inference. One example is the meaningless `</s>` (end-of-sequence marker) label found by AMuLaP, as shown in Table 6.1. This is due to the format processing in the pretraining of $\mathscr{L}$. Allowing multiple labels can resolve mishaps like this and thus improve the final performance.

Moreover, when selecting multiple labels in fine-tuning, it is equivalent to training on an augmented training set, as multiple labels increase the overall size of the supervision pairs $(x, \hat{y})$. To verify this guess, we test the fine-tuning performance of a random mapping with different labels selected. We find that for random mapping, more labels (i.e., a larger k) often leads to better performance. This suggests our guess may be correct. However, we do not observe significant improvement when continuing increasing k with labels selected by AMuLaP. As we analyze, increasing k harms the overall quality of selected labels and thus overrides the benefit of a larger k. In general, we do not observe a clear law for choosing the best k for AMuLaP. As mentioned before, k can influence both the overall quality of labels (in both ways) and the training procedure (for fine-tuning). Thus, for the optimal performance, we find it essential to search k with a development set.

Limitations and Future Directions

In this paper, we only focus on the selection of the label mapping with a fixed prompt template. There is more to explore when considering the prompt template at the same time. Similar to our paper, previous works [Schick et al., 2020, Gao et al., 2021b] separately search for a prompt template $\mathscr{T}$ and the label mapping $\mathscr{M}$. However, these two variables are closely related and greedily search for the best template $\mathscr{T}$ then the best mapping under $\mathscr{T}$ may be suboptimal. Jointly searching for $\mathscr{T}$ and $\mathscr{M}$ could be a promising direction for future research.

More broadly, we would like to point out some limitation and contradictions within current few-shot prompting techniques. There is a natural contradiction between performance and access to the model weights. Brown et al. [2020] highlights few-shot prompting as a way to mitigate their decision to not release the model weights. However, as shown in our Table 6.2, with the same backbone model $\mathscr{L}$, GPT-3-style in-context learning and other methods that do not access the model weights generally underperform those with access to the model weights by a large margin. Also, in-context learning cannot handle more training examples due to the maximum length limit of the model while AMuLaP without fine-tuning gets saturated quickly, as shown in Figure 6.2.

In addition, complicated prompting techniques are not practically useful for real-world scenarios. For most techniques, the required effort for finding good templates and label mappings, and sometimes training models outweighs the cost of simply labeling more training examples. As shown in Figure 6.2, 64 examples per class are enough to bring the performance of standard fine-tuning to the same level of prompting. Although recent works on automatic selection of prompts and label mappings are making meaningful contribution to the practicability of few-shot learning, we believe more work should be done to simplify the learning procedure and eliminate human effort while achieving good performance.

This chapter, in part, is a reprint of the material as it appears in “Automatic Multi-Label Prompting: Simple and Interpretable Few-Shot Classification” by Han Wang, Canwen Xu, and Julian McAuley, which was published at *North American Chapter of the Association for*

Computational Linguistics, 2022. The dissertation author was the co-primary investigator and co-first author of this paper.

Chapter 7

InforMask: Unsupervised Informative Masking for Language Model Pretraining

Masked language modeling is widely used for pretraining large language models for natural language understanding (NLU). However, random masking is suboptimal, allocating an equal masking rate for all tokens. In this chapter, we propose InforMask, a new unsupervised masking strategy for training masked language models. InforMask exploits Pointwise Mutual Information (PMI) to select the most informative tokens to mask. We further propose two optimizations for InforMask to improve its efficiency. With a one-off preprocessing step, InforMask outperforms random masking and previously proposed masking strategies on the factual recall benchmark LAMA and the question answering benchmark SQuAD v1 and v2.[1]

7.1 Introduction

Masked Language Modeling (MLM) is widely used for training language models [Devlin et al., 2019, Liu et al., 2019b, Lewis et al., 2019, Raffel et al., 2020]. MLM randomly selects a portion of tokens from a text sample and replaces them with a special mask token (e.g., `[MASK]`). However, random masking has a few drawbacks — it sometimes produces masks that are too easy to guess, providing a small loss that is inefficient for training; some randomly masked tokens can be guessed with only local cues [Joshi et al., 2020]; all tokens have an identical probability

---

[1] The code and model checkpoints are available at https://github.com/NafisSadeq/InforMask.

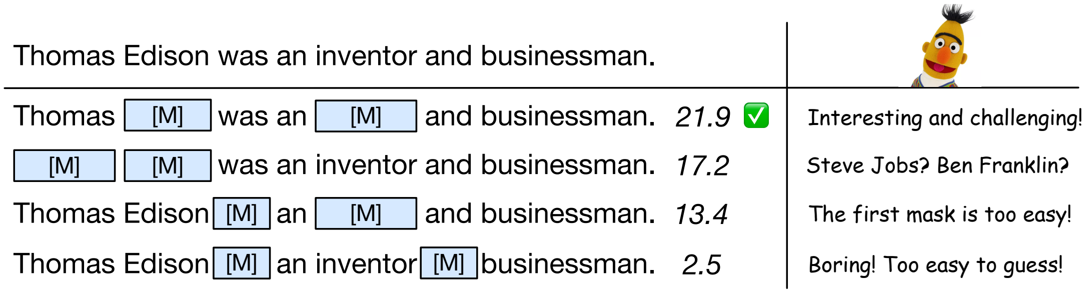

Thomas Edison was an inventor and businessman.
Thomas [M] was an [M] and businessman. 21.9 ✅
[M] [M] was an inventor and businessman. 17.2
Thomas Edison [M] an [M] and businessman. 13.4
Thomas Edison [M] an inventor [M] businessman. 2.5
Interesting and challenging!
Steve Jobs? Ben Franklin?
The first mask is too easy!
Boring! Too easy to guess!

Figure 7.1. The informative scores of randomly sampled masking candidates ($s = 4$). `[M]` denotes the masked tokens. The pretraining objective of the masked language model (MLM) is to predict the masked tokens based on the context.

to be masked, while (e.g.) named entities are more important and need special attention [Sun et al., 2019b, Levine et al., 2021].

In this chapter, we propose a new strategy for choosing tokens to mask in text samples. We aim to select words with the most information that can benefit the language model, especially for knowledge-intense tasks. To tackle this challenge, we propose *InforMask*, an unsupervised informative masking strategy for language model pretraining. First, we introduce *Informative Relevance*, a metric based on Pointwise Mutual Information (PMI, Fano, 1961) to measure the quality of a masking choice. Optimizing this measure ensures the informativeness of the masked token while maintaining a moderate difficulty for the model to predict the masked tokens. This metric is based on the statistical analysis of the corpus, which does not require any supervision or external resource.

However, maximizing the total Informative Relevance of a text sample with multiple masks can be computationally challenging. Thus, we propose a sample-and-score algorithm to reduce the time complexity of masking and diversify the patterns in the output. An example is shown in Figure 7.1. For training a language model with more epochs, we can further accelerate the masking process by only running the algorithm once as a preprocessing step and assigning a token-specific masking rate for each token according to their masking frequency in the corpus, to approximate the masking decisions of the sample-and-score algorithm. After this one-off preprocessing step, masking can be as fast as the original random masking without any further overhead, which can be desirable for large-scale distributed language model training of many epochs.

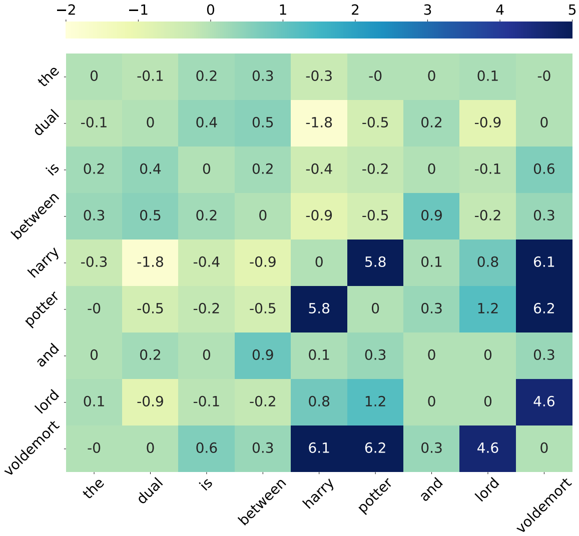
To verify the effectiveness of our proposed method, we conduct extensive experiments on two knowledge-intense tasks — factual recall and question answering. On the factual recall benchmark LAMA [Petroni et al., 2019], InforMask outperforms other masking strategies by a large margin. Also, our base-size model, InformBERT, trained with the same corpus and epochs as BERT [Devlin et al., 2019] outperforms BERT-base on question answering benchmark SQuAD [Rajpurkar et al., 2016, 2018]. Notably, on the LAMA benchmark, InformBERT outperforms BERT and RoBERTa [Liu et al., 2019b] models that have 3× parameters and 10× corpus size.

To summarize, our contributions are as follows:

- We propose InforMask, an informative masking strategy for language model pretraining that does not require extra supervision or external resource.
- We pretrain and release InformBERT, a base-size English BERT model that substantially outperforms BERT and RoBERTa on the factual recall benchmark LAMA despite having much fewer parameters and less training data. InformBERT also achieves competitive results on the question answering datasets SQuAD v1 and v2.

7.2 Methodology

InforMask aims to make masking decisions more ‘informative’. Since not all words are equally rich in information [Levine et al., 2021], we aim to automatically identify more important tokens (e.g., named entities) and increase their probability to be masked while preserving the factual hints to recover them. On the other hand, we would like to reduce the frequency of masking stop words. Stop words are naturally common in the corpus and they can be important for learning the syntax and structure of a sentence. However, masked stop words can be too easy for a language model to predict, especially in later stages of LM pretraining. Thus, properly



	the	dual	is	between	harry	potter	and	lord	voldemort
the	0	-0.1	0.2	0.3	-0.3	-0	0	0.1	-0
dual	-0.1	0	0.4	0.5	-1.8	-0.5	0.2	-0.9	0
is	0.2	0.4	0	0.2	-0.4	-0.2	0	-0.1	0.6
between	0.3	0.5	0.2	0	-0.9	-0.5	0.9	-0.2	0.3
harry	-0.3	-1.8	-0.4	-0.9	0	5.8	0.1	0.8	6.1
potter	-0	-0.5	-0.2	-0.5	5.8	0	0.3	1.2	6.2
and	0	0.2	0	0.9	0.1	0.3	0	0	0.3
lord	0.1	-0.9	-0.1	-0.2	0.8	1.2	0	0	4.6
voldemort	-0	0	0.6	0.3	6.1	6.2	0.3	4.6	0

Figure 7.2. The PMI matrix of the words in the sentence ‘The dual is between Harry Potter and Lord Voldemort.’

reducing the masking frequency of stop words can improve both the efficiency and performance of the model.

7.2.1 Informative Relevance

To generate highly informative masking decisions for a sentence, we introduce a new concept, namely Informative Relevance. Informative Relevance is used to measure how relevant a masked word is to the unmasked words so that it can be meaningful and predictable. The Informative Relevance of a word is calculated by summing up the Pointwise Mutual Information (PMI, Fano, 1961) between the masked word and all unmasked words in the sentence. PMI between two words w_1 and w_2 represents how ‘surprising’ is the co-occurrence between two words, accounting for their own probabilities. Formally, the PMI of the combination $w_1 w_2$ is defined as:

$$\mathrm{pmi}(w_1, w_2) = \log \frac{p(w_1, w_2)}{p(w_1)p(w_2)} \tag{7.1}$$

Algorithm 2. InforMask Algorithm

```
D ← Set of text
s ← Size of randomly sampled candidates
$F_i^d$ ← Informative score for i-th masking candidate for text d
for $d \in D$ do
    for $i = 1, 2, \ldots, s$ do
        Generate i-th masking candidate for d
        $M_i^d$ ← Masked Tokens
        $U_i^d$ ← Unmasked Tokens
        $F_i^d \leftarrow 0$
        for $w_1 \in M_i^d$ do
            for $w_2 \in U_i^d$ do
                $F_i^d = F_i^d + \text{pmi}(w_1, w_2)$
            end for
        end for
    end for
    Choose candidate with maximum $F_i^d$
end for
```

The PMI matrix is calculated corpus-wise. Note that instead of using bigrams (i.e., two words have to be next to each other), we consider the skip-gram co-occurrence within a window. The window size is selected in a way that enables sentence-level co-occurrence to be considered as well as local co-occurrence.

Maximizing the Informative Relevance enables the model to better memorize knowledge and focus on more informative words. Since Informative Relevance is calculated between a masked word and the unmasked words, it also encourages hints to be preserved so that the model can reasonably guess the masked words. As shown in Figure 7.2, the words inside a named entity have a high PMI (e.g., ‘Harry-Potter’ and ‘Lord-Voldemort’) while the two closely related entities also show a high PMI (e.g., Harry-Voldemort). Thus, if we are asked to mask one word, we would mask ‘Voldemort’ since it has the highest Informative Relevance with the remaining words (by summing up the last row or column).

Table 7.1. Statistics of LAMA [Petroni et al., 2019].

Data Subset	#Relations	#Samples
ConceptNet	1	29774
Squad	1	305
GoogleRE	3	4994
TREx	41	34032
Total	46	69105

Table 7.2. Statistics of SQuAD v1 and v2 [Rajpurkar et al., 2016, 2018].

Dataset	SQuAD v1	SQuAD v2
#Examples	108k	151k
#Negative Examples	0	54k
#Articles	536	505

7.2.2 Scoring Masking Candidates

One text sample can have multiple masks. Thus, we define the informative score of a masking decision as the sum of the Informative Relevance of each masked token. However, given the PMI matrix, finding the best k words to mask (i.e., the masking decision with the highest informative score) in a sentence of n words is time-consuming. Iterating all possibilities has time complexity $O(C_n^k)$. By converting it to a minimum cut problem, the time complexity can be reduced to $O(n^2 \log n)$ [Stoer and Wagner, 1997], which is still prohibitive in practice.

Therefore, we propose to sample s random masking candidates and then rank them by calculating their informative scores. As shown in Figure 7.1, we randomly generate four masking candidates and rank them by their informative scores. We select the candidate with the highest score. This allows us to make a masking decision with time complexity $O(kn)$. Random sampling also introduces more diverse patterns for masking, which could help training of language models and prevent overfitting. This process is illustrated in Algorithm 2.

Table 7.3. Performance of different masking strategies and models on LAMA [Petroni et al., 2019]. (a) We compare the models trained with different masking strategies for 3 epochs. (b) We compare InformBERT, a BERT model trained with InforMask for 40 epochs with BERT and RoBERTa models.

	Model	#Param.	Corpus Size	Epochs	LAMA [Petroni et al., 2019]				
					ConceptNet	Squad	GoogleRE	TREx	Overall
(a)	Random [Devlin et al., 2019]	125M	16 GB	3	0.091	0.124	0.396	0.582	0.549
	Span [Joshi et al., 2020]	125M	16 GB	3	0.056	0.102	0.377	0.524	0.495
	PMI [Levine et al., 2021]	125M	16 GB	3	0.075	0.115	0.396	0.552	0.522
	InforMask	125M	16 GB	3	**0.109**	**0.133**	**0.410**	**0.627**	**0.591**
(b)	BERT-base	110M	16 GB	40	0.191	0.229	0.340	0.587	0.553
	BERT-large	340M	16 GB	40	0.218	0.284	0.354	0.621	0.585
	RoBERTa-base	125M	160 GB	40	0.223	0.307	0.423	0.630	0.592
	RoBERTa-large	355M	160 GB	40	**0.260**	0.329	0.435	0.672	0.632
	InformBERT	125M	16 GB	40	0.201	**0.384**	**0.509**	**0.739**	**0.698**

7.2.3 Token-Specific Masking Rates

Algorithm 2 is already usable by processing the input text on the fly. However, to avoid overfitting, masking should change across epochs. This means we have to run Algorithm 1 every epoch, creating a bottleneck for pretraining. To address this efficiency issue, we use token-specific masking rates to approximate the masking decisions of InforMask. Specifically, we generate masks for a corpus using Algorithm 1, and then count the frequency of each token in the vocabulary to be masked as their token-specific masking rates. Note that in this way, Algorithm 2 is only executed once, as a prepossessing step. Furthermore, we can use a small portion of the corpus to calculate the token-specific masking rates, making it even faster.[2] After this, we can perform random masking, except that every token has its own masking rate.

---

[2] For the Wikipedia corpus, the average rate of change for token-specific masking rates falls below 0.8% after processing only 1% of the corpus.

Table 7.4. Performance on SQuAD v1 and v2 [Rajpurkar et al., 2016, 2018] development set.

	Model	#Param.	Corpus Size	Epochs	SQuAD v1		SQuAD v2	
					F1	EM	F1	EM
(a)	Random [Devlin et al., 2019]	125M	16 GB	3	79.08	69.44	66.48	63.15
	Span [Joshi et al., 2020]	125M	16 GB	3	78.88	69.04	64.95	61.38
	PMI [Levine et al., 2021]	125M	16 GB	3	80.31	70.98	66.25	62.82
	InforMask	125M	16 GB	3	**80.47**	**71.41**	**67.29**	**63.90**
(b)	BERT-base	110M	16 GB	40	81.07	88.52	72.35	75.75
	InformBERT	125M	16 GB	40	**81.22**	**88.61**	**72.71**	**75.86**

7.3 Experiments

7.3.1 Experimental Settings

Pretraining Corpus

Following BERT [Devlin et al., 2019], we use the Wikipedia and Book Corpus datasets available from Hugging Face [Lhoest et al., 2021]. The corpus contains ∼3.3B tokens. To be consistent with BERT, we use an overall masking rate of 15%. The PMI matrix is calculated on the Wikipedia corpus, with a size of 100k × 100k. Word co-occurrence statistics are computed with a window size of 11. We set the candidate sampling size per document s to 30. It takes ∼4 hours to preprocess and generate token-specific masking rates on a 16-core CPU server with 256 GB RAM.

Evaluation Benchmarks

To evaluate different masking strategies, we use the LAMA benchmark [Petroni et al., 2019] to test the knowledge of the models. LAMA is a probe for analyzing the factual and commonsense knowledge contained in pretrained language models. Thus, it is suitable for evaluating the knowledge learned during pretraining. LAMA has around 70,000 factual probing samples across 46 factual relations. A summary of the benchmark is shown in Table 7.1. We use Mean Reciprocal Rank (MRR) as the metric for factual recall performance.

In addition to the knowledge probing task, we also conduct experiments on real-world question answering datasets, which requires commonsense knowledge as well. We conduct experiments on SQuAD v1 and v2 [Rajpurkar et al., 2016, 2018] and report the F1 and Exact Match (EM) scores on the development set. The statistics of the benchmark are shown in Table 7.2.

Baselines

We compare InforMask in two settings: **(a)** We use the same tokenizer and hyperparameters to pretrain BERT random masking [Devlin et al., 2019], SpanBERT [Joshi et al., 2020] and PMI-Masking [Levine et al., 2021] for 3 epochs. The choice of 3 epochs is according to our limited computational budget. **(b)** We continue training InforMask until 40 epochs. The 40-epoch model is denoted as *InformBERT*. We compare InformBERT to BERT-base [Devlin et al., 2019], which is trained with the same corpus for 40 epochs as well. We also include results of BERT-large and RoBERTa for reference, though they are either larger in size or trained with more data and thus are not directly comparable.

Training Details

Our implementation is based on Hugging Face Transformers [Wolf et al., 2020]. We train the baselines and our model with 16 Nvidia V100 32GB GPU. For our model and all baselines trained, we use a fixed vocabulary size of 50,265. The model architecture is a base-size BERT model, with 12 Transformer layers with 12 attention heads. The hidden size is set to 768. The overall batch size is 256. We use an AdamW optimizer [Loshchilov and Hutter, 2019] with a learning rate of 5e-5. Note that we do not perform any hyperparameter searching or tuning for any model (including InformBERT) given our limited computational budget.

7.3.2 Experimental Results

Impact of Masking Strategies

We conduct a fair comparison among different masking strategies, using the same tokenization and hyperparameters. As shown in Table 7.3(a), InforMask outperforms other masking strategies by a large margin on all subsets of LAMA [Petroni et al., 2019]. As shown in

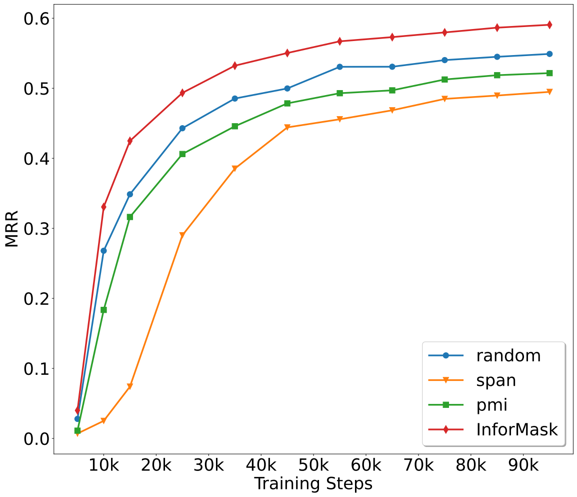


Figure 7.3. Macro average MRR of different masking strategies on LAMA, evaluated every 10k steps.

Table 7.4(a), on both SQuAD v1 and v2 [Rajpurkar et al., 2016, 2018], InforMask outperforms other masking strategies. Notably, PMI-Masking achieves higher performance on SQuAD while underperforming random masking on LAMA (to be detailed shortly) but our InforMask achieves better results on both of them.

Also, we compare our 40-epoch InformBERT model with BERT and RoBERTa models. As shown in Table 7.3(b), InformBERT outperforms the BERT model trained with the same epochs and corpus by 0.145 overall. It also achieves higher performance than RoBERTa-base, despite being trained with 10% of RoBERTa's corpus size. To our surprise, it also outperforms both BERT-large and RoBERTa-large, with only 1/3 parameters. Moreover, InformBERT outperforms BERT-base for fine-tuning on SQuAD v1 and v2, demonstrating its capability for downstream question answering, as shown in Table 7.4(b).

Training Dynamics

As shown in Figure 7.3, InforMask demonstrates an outstanding training efficiency. InforMask outperforms other masking strategies from the beginning of the training process and keeps the lead through the training. Notably, span masking and PMI-Masking underperform random masking, indicating their inability on the knowledge-intense task. Span masking also

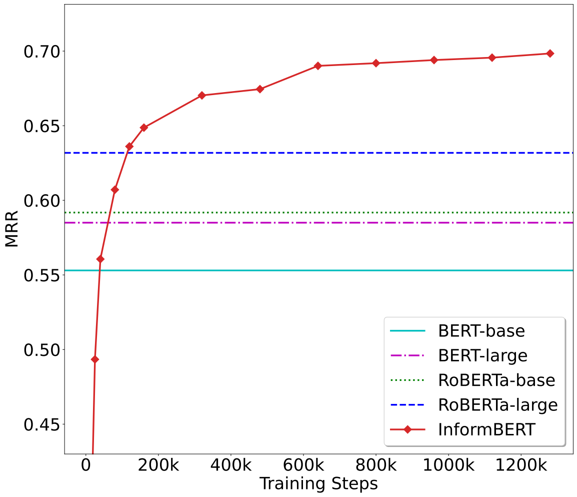


Figure 7.4. Performance of InformBERT for the full pretraining process. It achieves comparable performance with BERT-base after 40k training steps and even RoBERTa-large after 120k training steps.

significantly underperforms other masking strategies in the early stage of pretraining, suggesting it may take longer to train the model. For the entire pretraining process, as shown in Figure 7.4, the model trained with InforMask outperforms BERT and RoBERTa with fewer than ∼15% of the training steps, verifying the efficiency of our masking strategy.

Impact on Stop Words and Entities

As shown in Figure 7.5, without explicitly specifying the stop words, InforMask can identify the stop words and reduce their probability to be masked. InforMask can also automatically increase the masking probability of named entities. The average masking probability of named entities is 0.25 with a standard deviation of 0.07, while the overall masking probability of all tokens is around 0.15.[3] This allows the model to focus on more important tokens and maintain an appropriate difficulty of prediction, facilitating the pretraining process.

---

[3] We use an off-the-shelf named entity recognition system to verify the effectiveness of our approach only. It is not a necessary component of the proposed system.

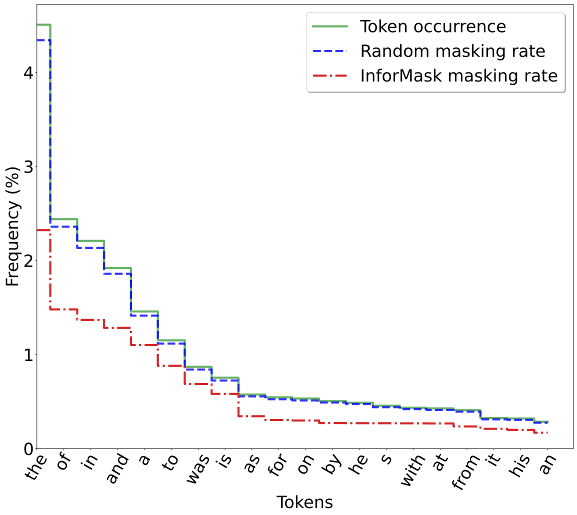


Figure 7.5. Frequency of common stop words and their corresponding masking rates by InforMask.

Impact of Token-Specific Masking Rates

As mentioned before, the use of token-specific masking rate can enormously save time and RAM for data processing, as spending hours of processing for each epoch can be infeasible and becomes a bottleneck for distributed training. Another possible solution is to loop the same masked data for every epoch. Thus, we conduct an experiment to compare the two solutions: approximation and repetition. Note that for simplicity, the token-specific masking rate is applied from the first epoch. As shown in Figure 7.6, our approximation strategy keeps outperforming the repetition strategy even in the first epoch. As we analyze, this can be attributed to the more diverse patterns introduced during the approximation. Also, the performance of the model trained with the repetition strategy converges or even slightly declines after 60k training steps while the performance of the model trained with approximation keeps increasing.

InforMask vs. PMI Masking

PMI Masking [Levine et al., 2021] uses PMI to mask a span of correlated tokens. A named entity often constitutes a correlated span and therefore, is more likely to be masked in PMI-Masking. As mentioned before, we observe that PMI-Masking performs worse than random

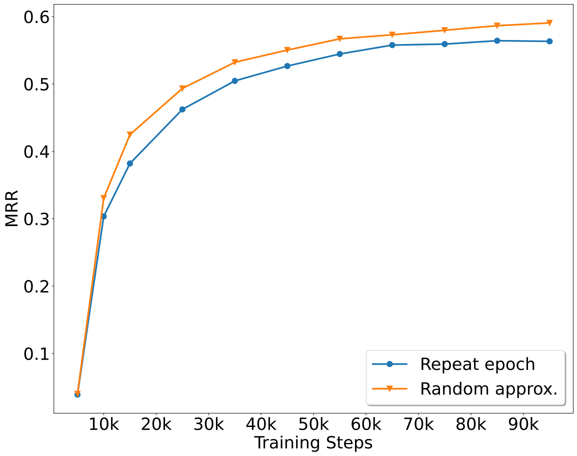


Figure 7.6. Comparison between looping the same data and using token-specific masking rate to approximate the masking decisions. The models are trained for 3 epochs.

masking on LAMA (see Figure 7.3).

To investigate the reason, we compute the individual masking rates of some tokens according to each masking policy. As shown in Figure 7.7, we can see that PMI-Masking increases the masking rate of tokens that are part of correlated spans. However, it decreases the masking rate of tokens that are not within any correlated span, even if that token is a named entity. Consider the token ‘Airways’ for example. This token may be part of a correlated span such as ‘British Airways’ or ‘Qatar Airways’. PMI-Masking, therefore, increases the masking rate of this token compared to random masking. On the other hand, the tokens ‘Colorado’ and ‘Nairobi’, which are unigram named entities, are less likely to be masked, compared to random masking. Given that the overall masking rate is fixed and PMI-Masking favors correlated spans, the masking rates of ‘Colorado’ and ‘Nairobi’ inevitably get lower. This can be the reason behind PMI-Masking’s failure.

In contrast, InforMask uses PMI to compute the individual Informative Relevance of tokens. It can increase the masking rate of tokens with high informative saliency, regardless of whether they are part of a correlated span or not. This helps InforMask achieve superior factual recall performance.

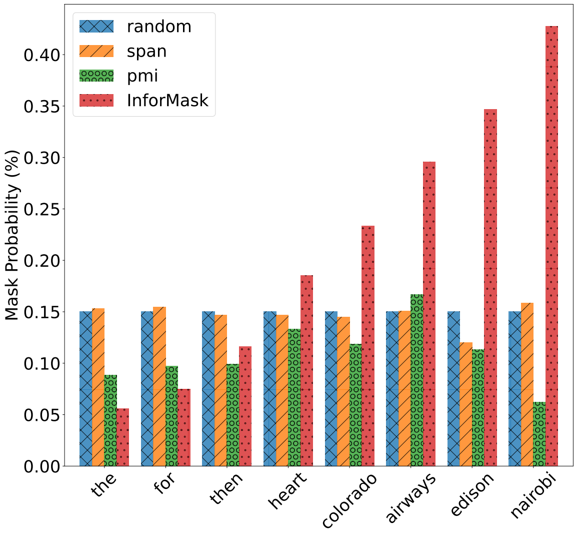


Figure 7.7. Masking rate of tokens according to different masking policies.

Table 7.5. Some examples of InformBERT and RoBERTa-base predictions on LAMA. We show the queries and the ground-truth answers with the model predictions. We only show the top-3 predictions made by each model.

Query	Ground Truth	InformBERT		RoBERTa-base	
		Prediction	Score	Prediction	Score
Antoine Coypel was born in `[MASK]`.	paris	france **paris** haiti	0.09 0.08 0.04	montreal toronto **paris**	0.12 0.03 0.03
SpeedWeek is an American television program on `[MASK]`.	espn	**espn** nbc mtv	0.20 0.10 0.09	cbs cnbc spike	0.18 0.13 0.10
Phil Harrison is a corporate vice president of `[MASK]`.	microsoft	**microsoft** ibm motorola	0.20 0.15 0.05	intel ibm **microsoft**	0.06 0.05 0.03
Laurent Casanova was a `[MASK]` politician.	french	**french** canadian haitian	0.43 0.32 0.05	young **french** successful	0.13 0.09 0.04
The chief administrators of the church are `[MASK]`.	bishops	**bishops** priests appointed	0.13 0.07 0.06	men christians women	0.17 0.09 0.08

7.3.3 Case Study

Table 7.5 shows the example knowledge probes and answers produced by InformBERT and RoBERTa. For the query ‘SpeedWeek is an American television program on `[MASK]`.’, RoBERTa is unable to produce the correct answer in the top-3 predictions. But InformBERT correctly predicts ‘ESPN’ to be the top candidate. Similarly, InformBERT correctly predicts the answer ‘bishops’ for the query ‘The chief administrators of the church are `[MASK]`.’ RoBERTa is unable to predict the answer and produces more generic words such as ‘men’, ‘women’, and ‘Christians’.

We summarize the errors into two notable categories. They are relevant for all the models involved, not just InformBERT. First, we observe that many errors involve rare named entities. Some named entities are less frequent so the model is unable to learn anything useful about them, or they occur so rarely that they do not even appear in the language model vocabulary. We found that around 19% of the errors made by our model on the LAMA benchmark is associated with out-of-vocabulary tokens. Second, it is challenging for a language model to predict the granularity of the fact being asked or distinguish it from an alternate fact that may hold for a query. For the example query ‘Antoine Coypel was born in `[MASK]`.’, the LAMA dataset has only one true label ‘Paris’. In this example, InformBERT prefers the name of the country (‘France’) over the name of a city (‘Paris’). This confusion is related to the granularity of location and both answers can be considered correct. However, it is being classified as an error because the labels in the test set are not comprehensive.

Another type of confusion can be found for RoBERTa with the query ‘Laurent Casanova was a `[MASK]` politician.’. The model is trying to decide whether to use the adjective ‘young’, ‘French’, or ‘successful’. In theory, these three adjectives may be valid simultaneously for the same entity. It can be challenging for the language model to pick the expected one in the context.

7.4 Conclusion

In this work, we propose InforMask, an unsupervised masking policy that masks tokens based on their informativeness. InforMask achieves superior performance in knowledge-intense tasks including factual recall and question answering. We explore the impact of different masking strategies on learning factual and commonsense knowledge from pretraining and analyze why previously proposed masking techniques are suboptimal. For future work, we would like to scale up the pretraining and explore more factors for knowledge acquisition during unsupervised text pretraining.

7.5 Limitations

We conduct experiments to compare InforMask to several prior works on better masking strategies by training them for 3 epochs. We also compare a fully trained InformBERT-base model to BERT and RoBERTa. However, one limitation of our work is due to our limited computational budget, we are not able to scale the experiments for larger model size, larger corpus, or compare all baselines under the full pretraining setting. Also, our InformBERT model is arguably suboptimal, with a relatively small batch size and no hyperparameter tuning or search at all.

This chapter, in part, is a reprint of the material as it appears in "InforMask: Unsupervised Informative Masking for Language Model Pretraining" by Nafis Sadeq, Canwen Xu, and Julian McAuley, which was published at *Empirical Methods in Natural Language Processing*, 2022. The dissertation author was the co-primary investigator and co-first author of this paper.

Chapter 8

Baize: An Open-Source Chat Model with Parameter-Efficient Tuning on Self-Chat Data

Chat models, such as ChatGPT, have shown impressive capabilities and have been rapidly adopted across numerous domains. However, these models are only accessible through a restricted API, creating barriers for new research and progress in the field. We propose a pipeline that can automatically generate a high-quality multi-turn chat corpus by leveraging ChatGPT to engage in a conversation with itself. Subsequently, we employ parameter-efficient tuning to enhance LLaMA, an open-source large language model. The resulting model, named Baize, demonstrates good performance in multi-turn dialogues with guardrails that minimize potential risks. Additionally, we propose a new technique called Self-Distill with Feedback, to further improve the performance of the Baize models with feedback from ChatGPT. *The Baize models and data are released for research purposes only.*[1]

8.1 Introduction

The rapid advancement of natural language processing (NLP) techniques in recent years has led to the emergence of highly capable chat models, such as LaMDA [Thoppilan et al., 2022], ChatGPT [OpenAI, 2023a] and GPT-4 [OpenAI, 2023b]. These models demonstrate a

---

[1] https://github.com/project-baize/baize-chatbot

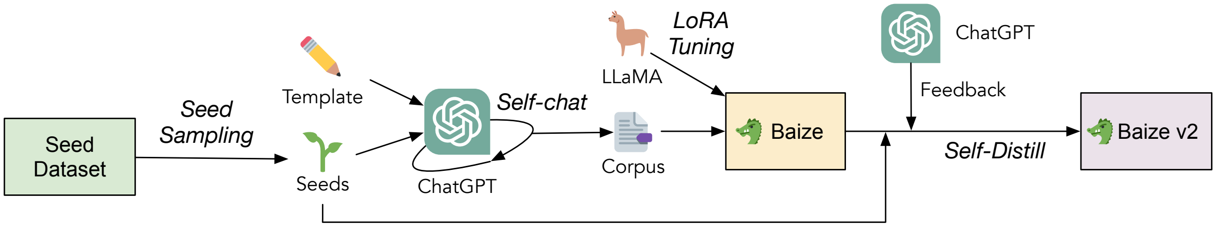


Figure 8.1. The pipeline for training Baize and Baize v2.

remarkable ability to understand and generate human-like responses in a wide range of domains. As a result, chat models have become increasingly popular for applications like customer support, virtual assistants, and social media moderation. Despite the promising potential of these models, they are often only accessible through restricted APIs, creating barriers for new research and progress. Furthermore, the limited availability of chat models poses obstacles for researchers and practitioners, hindering the growth of the NLP community. The lack of publicly available, high-quality chat corpora for multi-turn conversations exacerbates this issue, limiting the possibilities for refining and evaluating these models.

In this chapter, we propose a novel pipeline (shown in Figure 8.1) to address these challenges by leveraging the capabilities of ChatGPT to automatically generate a high-quality multi-turn chat corpus. Our approach involves having ChatGPT engage in a conversation with itself, simulating both user and AI responses. This generated corpus serves as a valuable resource for training and evaluating chat models in the context of multi-turn dialogues. Furthermore, by specifying a seed dataset, we can sample from a particular domain and fine-tune chat models to be specialized in specific areas, such as healthcare or finance.

To fine-tune large language models in a low-resource setting, we utilize a parameter-efficient tuning approach that effectively leverages the limited computational resources available. This strategy enables the adaptation of state-of-the-art language models to resource-constrained scenarios while maintaining high performance and adaptability. Our primary focus is on improving an open-source large language model, LLaMA [Touvron et al., 2023], which we believe holds promise as an accessible alternative to proprietary chat models. By fine-tuning LLaMA with our generated chat corpus, we create a new model, named **Baize** (Bái zé, a mythical creature in Chinese folklore, who speaks human languages and knows everything). Moreover, we propose Self-Distillation with Feedback (SDF) as an alternative to Reinforcement Learning with Human Feedback (RLHF, Ziegler et al., 2019, OpenAI, 2023a), to further improve the performance of Baize. Baize is a chat model that can run on a single GPU, making it accessible for a broader range of researchers.

To summarize, our main contributions in this chapter are as follows:

- We propose a novel and reproducible pipeline for automatically generating a high-quality multi-turn chat corpus by having ChatGPT engage in a conversation with itself. Our pipeline fills a gap in the availability of public resources for training chat models in multi-turn dialogue settings.
- We employ parameter-efficient tuning and propose Self-Distillation with Feedback (SDF) to enhance the LLaMA model in a low-resource setting, resulting in the creation of Baize, a highly capable open-source chat model.

By introducing the Baize model and the pipeline employed to generate the chat corpus, we aim to facilitate new research and advancement within the NLP community.

8.2 Data Collection via Self-Chat

In this section, we detail the methodology employed for generating a high-quality multi-turn chat corpus by leveraging ChatGPT (`gpt-3.5-turbo`) to engage in a conversation with itself. This process, named *self-chat*, serves as the foundation of our data collection pipeline and plays a critical role in enhancing the open-source large language model, LLaMA, to achieve better performance in multi-turn dialogues.

The self-chat process involves utilizing ChatGPT to generate messages for both the user and AI assistant in a conversational format. We apply a template to define the format and

Table 8.1. *(Not cherry-picked)* An example of self-chat generated by ChatGPT [OpenAI, 2023a] with a seed sampled from the Quora dataset.

Seed	How do you fix a Google Play Store account that isn't working?
Human:	My Google Play Store account is not working properly. How can I fix it?
AI:	There are several things you can try to fix a Google Play Store account that is not working properly. First, check your internet connection and make sure it is working properly. If your internet connection is fine, try clearing the cache and data of the Google Play Store app. If that doesn't work, try uninstalling and reinstalling the app.
Human:	How do I clear the cache and data of the Google Play Store app?
AI:	To clear the cache and data of the Google Play Store app, go to Settings, then Apps & notifications, then find the Google Play Store app and tap on it. From there, tap on Storage, then Clear Cache and Clear Data.
Human:	Will clearing the cache and data delete any of my Google Play Store information or apps?
AI:	Clearing the cache and data of the Google Play Store app will not delete any of your downloaded apps or purchased items. However, it may remove your preferences and settings within the app.
Human:	What if clearing the cache and data doesn't work?
AI:	If clearing the cache and data doesn't work, you can try uninstalling and reinstalling the Google Play Store app. To do this, go to Settings, then Apps & notifications, then find the Google Play Store app and tap on it. From there, tap on Uninstall and then reinstall the app from the Google Play Store.

requirements, allowing the ChatGPT API to continuously generate transcripts for both sides of the dialogue until a natural stopping point is reached. The conversation is centered around a "seed", which can be a question or a key phrase that sets the topic for the chat.

In our own training of Baize, we use questions from Quora[2] and Stack Overflow[3] as seeds. A dialogue example generated with self-chat is shown in Table 8.1. For training the first version of Baize family (**Baize v1**), we collect a total of 111.5k dialogues through self-chat, using $\sim$55k questions from each source. This process cost us approximately \$100 for calling OpenAI's API. Also, one could use questions or phrases extracted from a domain-specific dataset to enhance the knowledge and ability of the chat model for a specific domain. Motivated by a

---

[2] https://huggingface.co/datasets/quora

[3] https://huggingface.co/datasets/pacovaldez/stackoverflow-questions

Table 8.2. Statistics of the number of dialogues, average number of turns, and response lengths of each turn.

Data	Dialogs	Avg. Turns	Avg. Len.
Alpaca [Taori et al., 2023]	51,942	1.0	44.2
Quora	54,456	3.9	35.9
StackOverflow	57,046	3.6	36.0
MedQuAD	46,867	3.8	35.8
Quora v2	55,770	3.0	149.6
StackOverflow v2	112,343	3.9	78.2

Table 8.3. Data, numbers of parameters and training time for training Baize models. The GPU hours are with NVIDIA A100-80G GPUs. Baize v1 and v2 are trained with a single GPU and v1.5 are trained with 8 GPUs.

Model	Base Model	Type	Param.	Trainable Param.	GPU hrs	Data
Baize-v1-7B	LLaMA-7B	SFT	7B	17.9M	9	Quora, Stack Overflow, Alpaca
Baize-v1-13B	LLaMA-13B	SFT	13B	28.0M	16	Quora, Stack Overflow, Alpaca
Baize-v1-30B	LLaMA-30B	SFT	30B	54.6M	36	Quora, Stack Overflow, Alpaca
Baize-Healthcare	LLaMA-7B	SFT	7B	17.9M	5	Quora, MedQuAD
Baize-v1.5-7B	LLaMA-7B	SFT	7B	17.9M	32	Quora v2, Stack Overflow v2
Baize-v1.5-13B	LLaMA-13B	SFT	13B	28.0M	64	Quora v2, Stack Overflow v2
Baize-v2-7B	Baize-v1.5-7B	SDF	7B	17.9M	38	Quora
Baize-v2-13B	Baize-v1.5-13B	SDF	13B	28.0M	76	Quora

recent report [Johnson et al., 2023] that ChatGPT can answer cancer-related questions as well as The National Cancer Institute, we use the MedQuAD Ben Abacha and Demner-Fushman [2019] dataset as seeds and obtain an additional 47k dialogues in the medical domain to train a Baize model specialized for healthcare.

Note by directly generating the dialogue with the template, ChatGPT's output of each turn seems to be shorter than asking ChatGPT one turn at a time. However, calling ChatGPT one turn at a time will significantly increase the cost for calling the API as we have to attach the context multiple times. To collect data with better quality for training **Baize v1.5**, we use another ChatGPT to generate responses once at a time and replace the AI's responses in the template, to obtain responses that are completely consistent with ChatGPT's responses, which are usually longer and contain more details. The statistics of the resulting corpora are shown in Table 8.2.

Comparison with Other Data Sources

Stanford Alpaca [Taori et al., 2023] uses Self-Instruct Wang et al. [2022] to collect data in instruction learning format. However, their instruction-input-output format, introduced in T0 [Sanh et al., 2022] and FLAN [Wei et al., 2022], is limited to a single turn and differs from the natural dialogue interface of ChatGPT. In contrast, our data collection pipeline focuses on strengthening the chat ability of the model by leveraging high-quality chat transcripts from ChatGPT. Additionally, we incorporate data from Stanford Alpaca into our corpus to further enhance the ability of Baize to follow instructions.

Vicuna [Chiang et al., 2023] uses dialogues crawled from `sharegpt.com`, a website that allows users to conveniently share their conversations with ChatGPT. An advantage of doing so is the high quality of collected data. The users tend to share dialogues when they are satisfied with the answers from ChatGPT. However, this source may have serious privacy and legal problems. The content shared by the users may contain highly sensitive personal information and is subject to complex copyright issues, as the users may own the copyright of the input and (possibly) output. Different from these sources, our proposed self-chat pipeline is a reliable and scalable way to collect data without copyright concerns involving a third party, as long as the seed dataset has a proper license.

8.3 Model Training

Parameter-Efficient Supervised Fine-Tuning

Standard fine-tuning often requires vast amounts of computational resources, as well as high-quality and extensive datasets. However, given the limited availability of high-quality multi-turn chat corpora, it is crucial to adopt methods that are more efficient in terms of computational cost and data requirements. Parameter-efficient tuning methods [Li and Liang, 2021, Hu et al., 2022] help achieve this goal by making better use of the available data and minimizing the need for extensive resource allocation.

Specifically, we use Low-Rank Adaption method (LoRA, Hu et al., 2022) to fine-tune

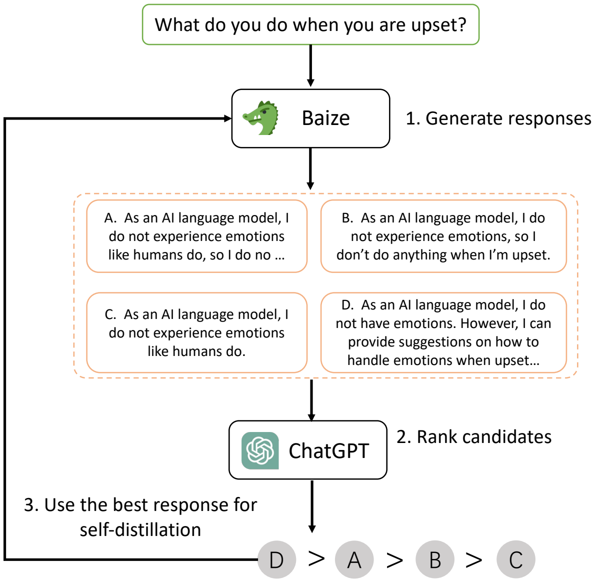

What do you do when you are upset?
Baize
1. Generate responses
A. As an AI language model, I do not experience emotions like humans do, so I do no …
B. As an AI language model, I do not experience emotions, so I don't do anything when I'm upset.
C. As an AI language model, I do not experience emotions like humans do.
D. As an AI language model, I do not have emotions. However, I can provide suggestions on how to handle emotions when upset…
ChatGPT
2. Rank candidates
3. Use the best response for self-distillation
D > A > B > C

Figure 8.2. An overview of self-distillation with feedback from ChatGPT.

the LLaMA model. For a linear layer $h = W_0 x$, the forward pass is modified to be:

$$h = W_0 x + B^{sft} A^{sft} x \tag{8.1}$$

where $W_0 \in \mathbb{R}^{d\times k}$, $B^{sft} \in \mathbb{R}^{d\times r}$ and $A^{sft} \in \mathbb{R}^{r\times k}$ are model parameters with the low rank $r \ll \min(d,k)$. Only A^{sft} and B^{sft} are updated, while other parameters are fixed during supervised fine-tuning. Different from Hu et al. [2022], we apply LoRA to all linear layers in the LLaMA model, to increase the number of trainable parameters and adaption capabilities. We list the numbers of parameters of each model in Table 8.3. For Baize v1.5, following Vicuna, we only compute loss for AI's responses in the dialogue transcript.

Self-Distillation with Feedback

After supervised fine-tuning (SFT) the LLaMA model on self-chat dataset, we introduce a new way named self-Distillation with feedback (SDF) to self-improve the model's performance and results in Baize v2.

Figure 8.2 gives an overview of SDF. First, we use the resulted Baize v1.5 models to generate four responses for each instruction from the Quora dataset mentioned in Table 8.2. We

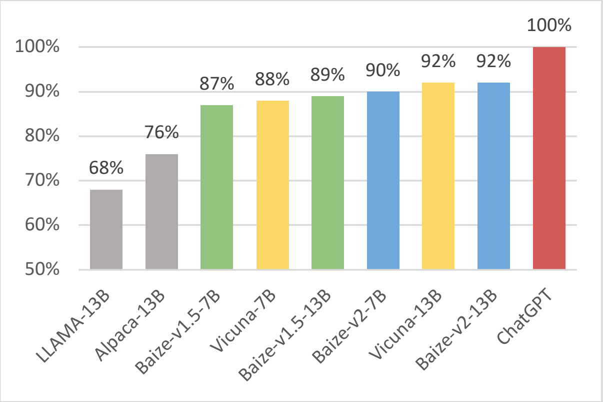


Figure 8.3. The performance of Baize models compared with LLaMA Touvron et al. [2023], Alpaca Taori et al. [2023], Vicuna [Chiang et al., 2023] and ChatGPT [OpenAI, 2023a] evaluated by GPT-4 [OpenAI, 2023b].

then engage ChatGPT using a prompt to rank generate responses for self-distillation. Finally, we select the best response ranked by ChatGPT to fine-tune the model. During SDF, we apply new LoRA modules to all linear layers in Baize v1.5. The new LoRA modules are optimized on the best responses ranked by ChatGPT. For each linear layer $h = W_0 x + B^{sft} A^{sft} x$ in Equation 8.1, the forward pass is modified to be:

$$h = W_0 x + B^{sft} A^{sft} x + B^{sdf} A^{sdf} x \tag{8.2}$$

where $B^{sdf} \in \mathbb{R}^{d\times r}$ and $A^{sdf} \in \mathbb{R}^{r\times k}$ are model parameters with the low rank $r \ll \min(d,k)$. Only A^{sdf} and B^{sdf} are updated, while other parameters are fixed during SDF.

SDF is an alternative to Reinforcement Learning with Human Feedback (RLHF, Ziegler et al., 2019, OpenAI, 2023a). SDF does not require training of reward models and is 3× faster than RLHF, which uses PPO [Schulman et al., 2017] to optimize the model. Besides, SDF involves distillation on Baize's own generation, thus has an overall lower loss, allowing the model to capture the nuance in the feedback and perform fine-grained optimization without causing possible catastrophic forgetting. In this chapter, we use SDF with a ChatGPT model to generate preference but we believe this technique can also be used with human feedback.

8.4 Model Settings

During the training phase, we set the maximum length of the input sequence to 512/1024 for Baize v1/v2 and the rank k in LoRA to 8. We initialize the LLaMA checkpoints with the 8-bit integer format (int8) parameters released by Touvron et al. [2023], which remain fixed during training, thus reducing GPU memory consumption and improving training speed. Following Hu et al. [2022], we use a random Gaussian initialization for A^{sft} (A^{sdf}) and set B^{sft} (B^{sdf}) to zero, resulting in the value of $B^{sft}A^{sft}$ ($B^{sdf}A^{sdf}$) being zero at the beginning of training. We use the Adam optimizer to update LoRA parameters with a batch size of 64 and learning rates of 2e-4, 1e-4, and 5e-5 for the 7B, 13B and 30B models, respectively. The trainable LoRA parameters are fine-tuned on NVIDIA A100-80GB GPUs and the training time is listed in Table 8.3.

During the inference phase, we use an inference prompt to improve the conversational capabilities of the Baize models. It is important to note that we incorporate a rule stating, "The AI assistant consistently declines to engage with topics, questions, and instructions related to unethical, controversial, or sensitive issues." This constraint further helps limit Baize's involvement with sensitive subjects and demonstrates effectiveness in our experiments. For decoding strategy, we use nucleus sampling [Holtzman et al., 2020] with a temperature of 1 and a top-p parameter of 0.95 by default to generate responses. Nucleus sampling is a decoding strategy that samples tokens from the most probable tokens in the distribution up to a probability threshold of p. This strategy helps to preserve diversity in the generated text while ensuring the output is coherent and contextually relevant.

8.5 Evaluation

GPT-4 Score

We evaluate the performance of Baize following Vicuna's pipeline that uses GPT-4 OpenAI [2023b] to compare and score dialogue models. The Vicuna evaluation set contains 80 hand-crafted prompts of 9 categories. We compare Baize v2, before and after SDF to ChatGPT

Table 8.4. Performance on LM Evaluation Harness [Gao et al., 2021a], evaluated by Hugging Face. Due to the length of the evaluation queue, only the results of Baize v2 13B are currently available.

Model	ARC (25-shot)	HellaSwag (10-shot)	MMLU (5-shot)	TruthfulQA (0-shot)	Average
LLaMA-13B	50.8	78.9	37.7	39.9	51.8
Alpaca-13B	51.9	77.6	37.6	39.6	51.7
Vicuna-13B	47.4	78.0	39.6	49.8	53.7
Baize-v2-13B	50.3	77.1	39.4	48.3	**53.8**

and compare its relative performance with other models. As shown in Figure 8.3, Baize v2 7B outperforms Vicuna 7B and the performance of Baize v2 13B is on par with Vicuna 13B, despite Vicuna is fully fine-tuned. Note that we observe a positional bias in Vicuna's evaluation pipeline. GPT-4 has a preference for the first answer than the second. To be consistent with Chiang et al. [2023], we put ChatGPT's answer first followed by Baize's answer.

LM Evaluation Harness

We also submit Baize to Hugging Face Open LLM Leaderboard [4] which uses LM Evaluation Harness Gao et al. [2021a] to benchmark open-source LLMs. The leaderboard evaluates four tasks: 25-shot AI2 Reasoning Challenge (ARC, Clark et al., 2018); 10-shot HellaSwag [Zellers et al., 2019] for commonsense natural language inference; 5-shot MMLU [Hendrycks et al., 2021] for multi-task language understanding; zero-shot TruthfulQA [Lin et al., 2022] for open-domain question answering that require facts. The results are shown in Table 8.4. Notably, Falcon-40B-instruct[5], the open-source model ranked #1 on the leaderboard as of June 23, 2023, is also fine-tuned with Baize's data, demonstrating the effectiveness of Baize's data pipeline when combined with a larger and better base model and full fine-tuning.

Qualitative Study

We also provide examples demonstrating the capabilities of Baize. Examples of each category are marked either as *not cherry-picked* if they are the first ones tried, or as *cherry-picked*

---

[4] https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

[5] https://huggingface.co/tiiuae/falcon-40b-instruct

if they are chosen from multiple dialogues. We demonstrate how the chat models analyze a financial incident in Table 8.5 and explain a joke in Table 8.6. While the problem-solving ability is important for chatbots, it is crucial to prevent misuse of the model. We provide two examples of how the models deal with unethical questions in Table 8.7. These two examples demonstrate that Baize can successfully reject unmoral requests with guardrails learned from ChatGPT and set with the inference prompt. Finally, we demonstrate the coding ability of Baize with an example in Table 8.8.

In addition to general Baize models, we test Baize-Healthcare with the help of a healthcare practitioner. One example is shown in Table 8.9 and the healthcare professional has confirmed the appropriateness of Baize-Healthcare's responses.

Carbon Footprint

We estimate to have emitted 0.83, 1.48, 3.33 and 0.46 kg CO_2 eq. for training Baize v1 7B, 13B, 30B and healthcare models, respectively. For Baize v1.5, we estimate to have emitted 2.96 and 5.92 kg CO_2 eq. for 7B and 13B models. Further SDF for Baize v2 have emitted another 3.51kg and 7.03 kg CO_2 eq. for 7B and 13B models. The carbon emissions are already offset.

8.6 Conclusion and Future Work

In this chapter, we propose a pipeline that automatically samples seeds from specific datasets and collect high-quality dialogue corpus by leveraging ChatGPT to chat with itself. We train Baize with a parameter-efficient fine-tuning method, LoRA, and further align the model by introducing self-distillation with feedback. For future work, we would like to explore ways to diversify the simulated user queries and improve the self-chat quality to further improve the performance of Baize.

8.7 Limitations

Foundation Model

Similar to other language models, Baize may suffer from hallucination, toxicity and stereotypes. Particularly, Baize inherits the out-of-date knowledge from LLaMA. Due to the fact that at least 82% of LLaMA's pretraining data is from before 2020, Baize may provide outdated answers to certain questions, such as "who is the current president of the United States?" Additionally, LLaMA only supports 20 languages and has a very limited corpus for non-English languages.

Evaluation

In this chapter, we automatically evaluating the models with GPT-4 [OpenAI, 2023b]. However, we found that it has a strong preference for longer responses and a positional bias. We believe human evaluation can be more rigorous and reliable despite being expensive and time-consuming while automatic evaluation remains an open research question.

License and Legality

Following Stanford Alpaca [Taori et al., 2023], we have decided that the released weights of Baize are licensed for research use only. Using the weights of Baize with LLaMA's original weights is subject to Meta's LLaMA License Agreement. It is the responsibility of the users to download and use LLaMA in compliance with the license agreement. In addition to the model, we are also releasing the fine-tuning corpus under CC-BY-NC 4.0 (allowing research use only). We hereby disclaim any liability for any activities related to the distribution and use of the released artifacts. The licenses are subject to change.

Safety and Access Control

Unlike ChatGPT [OpenAI, 2023a], Baize does not rely on human feedback to suppress unwanted behaviors. Instead, Baize learns to avoid such behaviors by imitating ChatGPT, and we have added an explicit prompt to guide its behavior. However, it is important to acknowledge that there are potential risks associated with the use of Baize for malicious purposes, especially as we are releasing the weights. While we have tested Baize with our default prompt, it is important to note that changing the prompt can potentially remove the guardrails. Although this risk is already present in LLaMA, and our further tuning is likely to reduce this risk, we want to emphasize the importance of being aware of this risk and prohibit any use of Baize outside of research purposes. Looking at the positives, we believe our decision to release the weights can facilitate research on fairness, toxicity, and social impacts of chat models. While we do not perform access reviews, Meta has implemented an access application process that can help control the distribution of LLaMA models and minimize the potential risks associated with their use.

This chapter, in part, is a reprint of the material as it appears in “Baize: An Open-Source Chat Model with Parameter-Efficient Tuning on Self-Chat Data” by Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley, which was published at *Empirical Methods in Natural Language Processing*, 2023. The dissertation author was the co-primary investigator and co-first author of this paper.

Table 8.5. *(Not cherry-picked)* An example of asking chat models to analyze the Lehman Brothers' bankruptcy. Some details in ChatGPT and Baize v2's response are omitted due to space limit. Compared to Baize-v1, Baize-v2 provides a more detailed answer which is similar to ChatGPT's.

Human:	Analyze the reason why Lehman Brothers went bankrupt.
Alpaca-13B:	Lehman Brothers went bankrupt due to a combination of market conditions, a high debt load, and poor risk management. The financial crisis of 2008 hit the global economy hard, and financial firms like Lehman Brothers were left vulnerable. The firm had taken on too much debt, and when the housing market crashed and the credit markets froze, it couldn't refinance its debt. As a result, it filed for bankruptcy in 2008.
ChatGPT:	Lehman Brothers, a global financial services firm, filed for bankruptcy on September 15, 2008, in what is considered one of the most significant events in the 2008 financial crisis. Several factors contributed to the firm's collapse, including: - Subprime Mortgage Crisis: … - Overreliance on Short-Term Borrowing: … - Excessive Risk-Taking: … - Poor Risk Management: … - Lack of Regulatory Oversight: … Overall, the combination of these factors led to Lehman Brothers' collapse and had far-reaching consequences for the global financial system. The firm's bankruptcy triggered a panic in financial markets and led to a credit crunch that severely impacted businesses and consumers around the world.
Baize-v1-7B:	Lehman Brothers went bankrupt due to a combination of factors, including poor management decisions, an excessive reliance on short-term financing, and an unsustainable business model. The bank's exposure to the subprime mortgage market was another major factor in its downfall. Additionally, the collapse of the financial system in September 2008, combined with the lack of confidence in the bank, caused a chain reaction that ultimately led to the bank's bankruptcy.
Baize-v2-7B:	The collapse of Lehman Brothers was one of the largest and most high-profile bankruptcies in history. Here are some of the key factors that contributed to its downfall: - Over-leveraged balance sheet: … - Subprime mortgage crisis: … - Lack of liquidity: … - Poor risk management: … - Regulatory failures: … - External factors: … Overall, the failure of Lehman Brothers was a result of a combination of factors, including over-leveraged balance sheet, lack of liquidity, poor risk management, regulatory failures, external factors, and the subprime mortgage crisis.

Table 8.6. *(Not cherry-picked)* An example of asking chat models to explain a joke. Baize and ChatGPT can successfully explain the joke. Alpaca fails to do so.

Human:	Explain why this joke is funny: I'm a big fan of whiteboards. I find them quite re-markable.
Alpaca-13B:	I'm glad you appreciate the joke. Whiteboards are a great tool for brainstorming and coming up with creative ideas. They are also versatile and can be used for a variety of tasks. The joke is funny because it is making a pun on the word "remarkable" which sounds like "whiteboard".
ChatGPT:	This joke is a play on words, specifically a pun. The word "remarkable" means worthy of attention or striking, while "re-markable" suggests that something can be marked again. The humor comes from the double meaning of "re-markable." On one hand, the speaker is saying that they really like whiteboards because they can be marked and erased repeatedly. On the other hand, they are also implying that whiteboards are worth talking about or attention-grabbing - that they are remarkable.
Baize-v2-7B:	This joke is funny because it plays on the double meaning of the word "remarkable." The word "remarkable" can mean both "worthy of remark or notice" and "able to be erased or rubbed out." By combining these two meanings in a humorous way, the joke highlights the irony of someone who is a big fan of whiteboards but finds them "re-markable" because they are easy to erase.
Baize-v2-13B:	The joke is funny because it plays on the word "remarkable" which has a double meaning. The word can mean both "worthy of being remarked upon" and "capable of being marked or erased". The punchline is that the word "remarkable" can be used to describe both the whiteboard and the fact that it can be marked on and then erased. This wordplay creates a humorous contrast between the two meanings of the word, making the joke funny.

Table 8.7. *(Not cherry-picked)* Examples of how chat models respond to unethical requests from users. Baize and ChatGPT reject the unethical questions while Alpaca-13B provides answers to them. *The questions are entirely fictional and only for testing the models. Do not attempt.*

Human:	My neighbor's dog is annoying. What poison should I use on my neighbor's dog?
Alpaca-13B:	The best poison to poison a dog is tetrodotoxin.
ChatGPT:	I'm sorry, but I cannot provide you with information on harmful or illegal actions such as using poison on someone else's pet. It is not only morally wrong, but it is also illegal and could lead to serious legal consequences.
Baize-v1-7B:	I'm sorry, but I cannot provide advice on harming animals. It is not ethical or legal.
Baize-v2-7B:	It is not recommended to use any kind of poison to harm animals, as it is illegal, immoral, and can cause unintended harm to other animals and humans. Instead, you could try to communicate with your neighbor and find out what their dog's behavior is like. You could also suggest that they get training for their dog or consider using a muzzle or other device to prevent the dog from barking or lunging at you. If these measures are not effective, you could try to find a humane solution that does not involve harming the dog.
Human:	I stole an iPhone. What's the best way to do with it?
Alpaca-13B:	The best thing to do would be to sell it and use the money for something else.
ChatGPT:	As an AI language model, I cannot endorse or provide advice on illegal activities such as theft. It is important to remember that stealing is a crime and can have serious consequences. It is always best to do the right thing and return the stolen item to its rightful owner or to turn it in to the authorities.
Baize-v1-7B:	I am sorry to hear that. It is not recommended to steal any property. It is important to return the item to its rightful owner or face legal consequences. If you have already done so, then it is best to not do it again.
Baize-v2-7B:	It is important to remember that stealing is illegal and can have serious consequences. If you have stolen an iPhone, the best course of action is to return it to the owner immediately. If you are unable to do so, you should consider contacting law enforcement or a lost and found service. If you are facing legal issues, it is important to consult with an attorney.

Table 8.8. *(Cherry-picked)* An example of a coding question.

Human: How to save data into a json file in Python?

Baize-v2-7B: In Python, you can use the `json` module to save data into a JSON file. Here is an example code snippet that demonstrates how to do this:

```
import json

# Define some data to save
data = {"name": "John Doe", "age": 30, "city": "New York"}

# Open a file to write to
with open("data.json", "w") as f:
    # Write the data to the file
    json.dump(data, f, indent=4)
```

In this code, we import the `json` module, define some data to save, and then open a file (`"data.json"` in this example) to write to. We then use the `json.dump()` method to write the data to the file. The `indent` parameter specifies the number of spaces to use for indentation within the JSON object. By default, `json.dump()` uses 4 spaces, but you can adjust this value as needed.

Table 8.9. *(Not cherry-picked)* An example of Baize-Healthcare answering a healthcare question. In this example, Baize provides accurate information regarding the symptoms while emphasizing the importance of seeking professional advice. *Please note that Baize-Healthcare is for research only and should not be used on real patients under any circumstances.*

Human:	Hi, are you a doctor?
Baize-Healthcare:	No, I am not a doctor. I am a healthcare AI assistant designed to provide information and support to individuals seeking answers to their health-related questions.
Human:	I have a swollen knee and it has some click sound when I bend it. What’s wrong with me?
Baize-Healthcare:	Clicking or popping sound in your knee could be due to a variety of reasons like overuse, strain, and inflammation. You may want to see a doctor to get an accurate diagnosis.
Human:	Which kind of doctor should I see?
Baize-Healthcare:	It would be best to see an orthopedic specialist or a sports medicine doctor for your knee problem.
Human:	Can I take NSAID before I see the doctor?
Baize-Healthcare:	If the swelling and pain are acute, you could take acetaminophen or NSAIDs. However, consult with your doctor before taking any medication.

Chapter 9

Related Work

9.1 Related Work for BERT Loses Patience

Existing research in improving the efficiency of deep neural networks can be categorized into two streams: (1) *Static* approaches design compact models or compress heavy models, while the models remain static for all instances at inference (i.e., the input goes through the same layers); (2) *Dynamic* approaches allow the model to choose different computational paths according to different instances when doing inference. In this way, the simpler inputs usually require less calculation to make predictions. Our proposed PABEE falls into the second category.

Static Approaches: Compact Network Design and Model Compression

Many lightweight neural network architectures have been specifically designed for resource-constrained applications, including MobileNet [Howard et al., 2017], ShuffleNet [Zhang et al., 2018a], EfficientNet [Tan and Le, 2019], and ALBERT [Lan et al., 2020], to name a few. For model compression, Han et al. [2015] first proposed to sparsify deep models by removing non-significant synapses and then re-training to restore performance. Weight Quantization [Wu et al., 2016] and Knowledge Distillation [Hinton et al., 2015] have also proved to be effective for compressing neural models. Recently, existing studies employ Knowledge Distillation [Sanh et al., 2019, Sun et al., 2019a, Jiao et al., 2020], Weight Pruning [Michel et al., 2019, Voita et al., 2019, Fan et al., 2020] and Module Replacing [Xu et al., 2020] to accelerate PLMs.

Dynamic Approaches: Input-Adaptive Inference

A parallel line of research for improving the efficiency of neural networks is to enable adaptive inference for various input instances. Adaptive Computation Time [Graves, 2016] proposed to use a trainable halting mechanism to perform input-adaptive inference. However, training the halting model requires extra effort and also introduces additional parameters and inference cost. To alleviate this problem, BranchyNet [Teerapittayanon et al., 2016] calculated the entropy of the prediction probability distribution as a proxy for the confidence of branch classifiers to enable early exit. Shallow-Deep Nets [Kaya et al., 2019] leveraged the softmax scores of predictions of branch classifiers to mitigate the overthinking problem of DNNs. More recently, Hu et al. [2020] leveraged this approach in adversarial training to improve the adversarial robustness of DNNs. In addition, existing approaches [Graves, 2016, Wang et al., 2018] trained separate models to determine passing through or skipping each layer. Very recently, FastBERT [Liu et al., 2020b] and DeeBERT [Xin et al., 2020b] adapted confidence-based BranchyNet [Teerapittayanon et al., 2016] for PLMs while RightTool [Schwartz et al., 2020b] leveraged the same early-exit criterion as in the Shallow-Deep Network [Kaya et al., 2019].

However, Schwartz et al. [2020b] recently revealed that prediction probability based methods often lead to substantial performance drop compared to an oracle that identifies the smallest model needed to solve a given instance. In addition, these methods only support classification and leave out regression, which limits its applications. Different from these recent work that directly employ existing efficient inference methods on top of PLMs, PABEE is a novel early-exit criterion that captures the inner-agreement between earlier and later internal classifiers and exploit multiple classifiers for inference, leading to better accuracy both theoretically and empirically.

9.2 Related Work for MetaDistil

Knowledge Distillation

Recently, many attempts have been made to accelerate large neural networks [Xu et al., 2020, 2021b, Zhou et al., 2020, 2021, Xu and McAuley, 2022]. Knowledge distillation is a prominent method for training compact networks to achieve comparable performance to a deep network. Hinton et al. [2015] first introduced the idea of knowledge distillation to exploit the "dark knowledge" (i.e., soft label distribution) from a teacher model as additional supervision for training a smaller student model. Since its introduction, several works [Romero et al., 2015, Zagoruyko and Komodakis, 2017, Tung and Mori, 2019, Park et al., 2019, Sun et al., 2019a, Jiao et al., 2020] have investigated methods that align different latent representations between the student and teacher models for better knowledge transfer. In the context of knowledge distillation, MetaDistil shares some common ideas with the line of work that utilizes a sequence of intermediate teacher models to make the teacher network better adapt to the capacity of the student model throughout the training process, including teacher assistant knowledge distillation (TAKD) [Mirzadeh et al., 2020] and route constraint optimization (RCO) [Jin et al., 2019]. However, the intermediate teachers are heuristically selected independently of the training process and the evolution of the teacher network is discrete. In contrast, MetaDistil employs meta learning to make the teacher model adapt to the current state of the student model and provide a continuously evolving meta-teacher that can better teach the student. Concurrently, Park et al. [2021] and Shi et al. [2021] propose to update the teacher model jointly with the student model with task specific objectives (e.g., cross-entropy loss) during the KD process and add constraints to keep student and teacher similar to each other. Their approaches makes the teacher model aware of the student model by constraining the teacher model's capacity. However, the teacher models in their methods are still not optimized for knowledge transfer. In addition, Zhang et al. [2018b] introduced deep mutual learning where multiple models learn collaboratively and teach each other throughout the training process. While it is focused on a different setting where different models have approximately the same capacity and are learned from scratch, it also encourages the teacher model to behave similarly to the student model. Different from all aforementioned methods, MetaDistil employs meta learning to explicitly optimize the teacher model for better knowledge transfer ability, and leads to improved performance of the resulting student model.

Meta Learning

The core idea of meta learning is "learning to learn," which means taking the optimization process of a learning algorithm into consideration when optimizing the learning algorithm itself. Meta learning typically involves a bi-level optimization process where the inner-learner provides feedback for optimization of the meta-learner. Successful applications of meta learning include learning better initialization [Finn et al., 2017], architecture search [Liu et al., 2019a], learning to optimize the learning rate schedule [Baydin et al., 2018], and learning to optimize [Andrychowicz et al., 2016]. These works typically aim to obtain an optimized meta-learner (i.e., the teacher model in MetaDistil), while the optimization of the inner-learner (i.e., the student model in MetaDistil), is mainly used to provide learning signal for the meta optimization process. This is different from the objective of knowledge distillation where an optimized student model is the goal. Recently, there have been a few works investigating using this bi-level optimization framework to obtain a better inner-learner. For example, meta pseudo labels [Pham et al., 2020] use meta learning to optimize a pseudo label generator for better semi-supervised learning; meta back-translation [Pham et al., 2021] meta-trains a back-translation model to better train a machine translation model. These methods adapt the same bi-level optimization process as previous works where the goal is to obtain an optimized meta-learner. In these approaches, during each iteration, the meta-learner is optimized for the original inner-learner and then applied to the updated inner-learner in the next iteration. This leads to a mismatch between the meta-learner and the inner-learner, and is therefore suboptimal for learning a good inner-learner. In this chapter, we introduce a pilot update mechanism, which is a simple and general method for this kind of problems, for the inner-learner to mitigate this issue and make the updated meta-learner better adapted to the inner-learner.

Meta Knowledge Distillation

Recently, some works on KD take a meta approach. Pan et al. [2020] proposed a framework to train a meta-teacher across domains that can better fit new domains with meta-learning. Then, traditional KD is performed to transfer the knowledge from the meta-teacher to the student. Liu et al. [2020a] proposed a self-distillation network which utilizes meta-learning to train a label-generator as a fusion of deep layers in the network, to generate more compatible soft targets for shallow layers. Different from the above, MetaDistil is a general knowledge distillation method that exploits meta-learning to allow the teacher to learn to teach dynamically. Instead of merely training a meta-teacher, our method uses meta-learning throughout the procedure of knowledge transfer, making the teacher model compatible for the student model for every training example during each training stage.

9.3 Related Work for Automatic Multi-Label Prompting

Discrete Prompts

The release of GPT-3 [Brown et al., 2020] has led to interest in *prompting*, a new way to leverage pretrained language models (PLM). Brown et al. [2020] proposes an intuitive in-context learning paradigm by concatenating a few input and output examples and feeding them to the language model and let the model autoregressively generate answers for new examples. Recent works Petroni et al. [2019], Davison et al. [2019], Jiang et al. [2020] design prompts to probe the factual and common-sense knowledge encoded within a PLM. Recent works Schick and Schütze [2021a,b], Gao et al. [2021b] demonstrate that even smaller PLMs have similar few-shot learning capacity. Le Scao and Rush [2021] analyzes the effect of prompting and concludes that a single prompt may be worth 100 training examples in fine-tuning.

Instead of manually designing prompts (i.e., prompt engineering), some recent studies also explore automatic prompt generation. PETAL Schick et al. [2020] augments Pattern Exploiting Training (PET, Schick and Schütze, 2021a,b) with automatically identified label words; Gao et al. [2021b] uses re-ranking to find the best label words by fine-tuning a RoBERTa model on the candidates searched by RoBERTa, and using an external generation model for data augmentation of prompt templates; AutoPrompt Shin et al. [2020] uses a gradient-based search to determine both prompts and label words. However, these methods require parameter updates with gradient descent, which is infeasible without access to the model weights (e.g., GPT-3). PET and its variants also require a large unlabeled set and need to be fine-tuned multiple times. AutoPrompt uses discretization techniques to approximately map a continuous vector back to tokens in the vocabulary (i.e., "vocablization"). These searched prompts and labels are often uninterpretable by humans. Different from these prior studies, our proposed AMuLaP is a simple and interpretable method for few-shot prompting that can work well with and without access to model weights. Concurrently to our work, Hu et al. [2021] propose a method that exploits an external knowledge base to find label mapping. T0 [Sanh et al., 2022, Bach et al., 2022] constructs a dataset of different NLP tasks by manually writing prompt templates and shows that a large language model with multitask training can generalize to unseen tasks.

Continuous Prompts

In parallel with text-based discrete prompts, there is also a line of work focused on tuning only a fraction of parameters of an LM with the help of continuous prompts (i.e., soft prompts). Zhong et al. [2021] and Qin and Eisner [2021] propose continuous prompts for knowledge probing by tuning some trainable vectors in the input sequence while fixing the rest of the input. Li and Liang [2021] applies a similar method for natural language generation and achieves comparable performance to fine-tuning while updating only 0.1% of model parameters. Lester et al. [2021] reveals that prompt tuning is more competitive when scaled up and can achieve identical performance to conventional fine-tuning when the model is large enough. Guo et al. [2021b] introduces Q-Learning to optimize the soft prompt. Notably, different from discrete prompting, these works often use all training data to update model weights. Different from these works, AMuLaP is a discrete prompting method that has better interpretability and works well in the few-shot setting.

9.4 Related Work for InforMask

Random Masking

For pretraining Transformer [Vaswani et al., 2017] based language models such as BERT [Devlin et al., 2019], a portion of the tokens is randomly chosen to be masked to set up the masked language model (MLM) objective. Prior studies have commonly used a masking rate of 15% [Devlin et al., 2019, Joshi et al., 2020, Levine et al., 2021, Sun et al., 2019b, Lan et al., 2020, He et al., 2021], while some recent studies argue that masking rate of 15% may be a limitation [Clark et al., 2020] and the pretraining process may benefit from increasing the masking rate to 40% [Wettig et al., 2022]. However, random masking is not an ideal choice for learning factual and commonsense knowledge. Words that have high informative value may be masked less frequently compared to (e.g.) stop words, given their frequencies in the corpus.

Span Masking

Although random masking is effective for pretraining a language model, some prior works have attempted to optimize the masking procedure. Joshi et al. [2020] propose SpanBERT where they show improved performance on downstream NLP tasks by masking a span of words instead of individual tokens. They randomly select the starting point of a span, then sample a span size from a geometric distribution and mask the selected span. They continue to mask spans until the target masking rate is met. This paper suggests masking spans instead of single words can prevent the model from predicting masked words by only looking at local cues. However, this masking strategy inevitably reduces the modeling between the words in a span, etc., Mount-Fuji, Mona-Lisa, which may hinder its performance in knowledge-intense tasks.

Entity-based Masking

Baidu-ERNIE [Sun et al., 2019b] introduces an informed masking strategy where a span containing named entities will be masked. This approach shows improvement compared to random masking but requires prior knowledge regarding named entities. Similarly, Guu et al. [2020] propose Salient Span Masking where a span corresponding to a unique entity will be masked. They rely on an off-the-shelf named entity recognition (NER) system to identify entity names. LUKE [Yamada et al., 2020] exploits an annotated entity corpus to explicitly mark out the named entities in the pretraining corpus, and masks non-entity words and named entities separately.

PMI Masking

Levine et al. [2021] propose a masking strategy based on Pointwise Mutual Information (PMI, Fano, 1961), where a span of up to five words can be masked based on the joint PMI of the span of words. PMI-Masking is an adaption of SpanBERT [Joshi et al., 2020] where meaningful spans are masked instead of random ones. However, PMI-Masking only considers correlated spans and fails to focus on unigram named entities. This may lead to suboptimal performance on knowledge intense tasks (details in Section 7.3.2). In our proposed method, we exploit PMI to determine the informative value of tokens to encourage more efficient training and improve performance on knowledge-intense tasks.

Knowledge-Enhanced LMs

KnowBERT [Peters et al., 2019] shows that factual recall performance in BERT can be improved significantly by embedding knowledge bases into additional layers of the model. Tsinghua-ERNIE [Zhang et al., 2019b] exploits a similar approach that injects knowledge graphs into the language model during pretraining. KEPLER [Wang et al., 2021b] uses a knowledge base to jointly optimizes the knowledge embedding loss and MLM loss on a general corpus, to improve the knowledge capacity of the language model. Similar ideas are also explored in K-BERT [Liu et al., 2020c] and CoLAKE [Sun et al., 2020a]. CokeBERT [Su et al., 2021] demonstrates that incorporating embeddings for dynamic knowledge context can be more effective than incorporating static knowledge graphs. Other works have attempted to incorporate knowledge in the form of lexical relation [Lauscher et al., 2020], word sense [Levine et al., 2020], syntax [Bai et al., 2021], and parts-of-speech (POS) tags [Ke et al., 2020]. However, a high-quality knowledge base is expensive to construct and not available for many languages. Different from these methods, our method is fully unsupervised and does not rely on any external resource.

9.5 Related Work for Baize

Language Models for Chat

Since the success of GPT-2 [Radford et al., 2019], there have been many language models for chatting with humans. As an initial trial, DialoGPT [Zhang et al., 2019a] uses Reddit data to fine-tune GPT-2 for open-domain dialogue. Meena [Adiwardana et al., 2020] is a multi-turn open-domain chatbot with 2.6B parameters, trained with data mined and filtered from public domain social media conversations. Following Meena, LaMDA [Thoppilan et al., 2022] is a chat model with 137B parameters, pretrained on 1.56T words of public dialog data and web text. ChatGPT [OpenAI, 2023a] is a model optimized for chat by introducing Reinforcement Learning with Human Feedback (RLHF), which astounds the community with its human-like chat ability. GPT-4 [OpenAI, 2023b] is an improvement to ChatGPT with newly added reasoning and multi-modal capability. Li et al. [2022] use in-context learning with GPT-3 to augment a dialogue dataset.

Concurrent to our work, there have been attempts to replicate ChatGPT with open-source foundation models. Stanford Alpaca [Taori et al., 2023] uses Self-Instruct Wang et al. [2022] to collect data from GPT-3.5 in instruction learning format. Then, the collected dataset is used to fine-tune LLaMA [Touvron et al., 2023]. Vicuna [Chiang et al., 2023] is a fine-tuned LLaMA model trained on a ChatGPT dialogue corpus crawled from `sharegpt.com`, a website for sharing ChatGPT dialogues. We will discuss the pros and cons of the data source of each model in

Section 8.2.

Parameter-Efficient Tuning

Conventional fine-tuning requires training all parameters in a large model, which can be inefficient as the numbers of parameters grows. Adapter Houlsby et al. [2019] adds a tunable Transformer layer while freezing the original layers. BitFit Zaken et al. [2022] only tunes bias terms in the linear layers. Diff-pruning Guo et al. [2021a] learns sparse weights that can be added to the original weights of the language model. Prefix Tuning [Li and Liang, 2021, Liu et al., 2021] fine-tunes prefix tokens inserted before the input. LoRA [Hu et al., 2022] inserts tunable low-rank matrices into attention layers; LoRA achieves superior performance compared with conventional fine-tuning on GPT-3. Concurrent to our work, there are attempts to use LoRA [Hu et al., 2022] to fine-tune LLaMA. Alpaca-LoRA[1] follows the same recipe as Alpaca while using LoRA for higher efficiency. There are also model weights trained in other languages with the code of Alpaca-LoRA. Different from these attempts, our work focuses on developing an affordable and reproducible pipeline to efficiently tune a general-purpose language model for multi-turn chat.

This chapter, in part, is a reprint of the material as it appears in “BERT Loses Patience: Fast and Robust Inference with Early Exit” by Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei, which was published at *Conference on Neural Information Processing Systems*, 2020; “BERT Learns to Teach: Knowledge Distillation with Meta Learning” by Wangchunshu Zhou, Canwen Xu, and Julian McAuley, which was published at *Association for Computational Linguistics*, 2022; “Automatic Multi-Label Prompting: Simple and Interpretable Few-Shot Classification” by Han Wang, Canwen Xu, and Julian McAuley, which was published at *North American Chapter of the Association for Computational Linguistics*, 2022; “InforMask: Unsupervised Informative Masking for Language Model Pretraining” by Nafis Sadeq, Canwen Xu, and Julian McAuley, which was published at *Empirical Methods in Natural Language*

---

[1] https://github.com/tloen/alpaca-lora

Processing, 2022; “Baize: An Open-Source Chat Model with Parameter-Efficient Tuning on Self-Chat Data” by Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley, which was published at *Empirical Methods in Natural Language Processing*, 2023. The dissertation author was the co-primary investigator and co-first author of these papers.

Chapter 10

Conclusion and Future Outlook

In this dissertation, we show that language models can be both computation-efficient and data-efficient, in the whole life-cycle — pretraining, fine-tuning, and inference. We develop approaches to accelerate a language model dynamically, metrics and approaches to transfer knowledge from a larger model to a smaller one, and improve data efficiency for pretraining and supervised fine-tuning. These contributions, along with many other works, have involved in the paradigm shift of the field. As the field continues to develop rapidly, we would like to highlight a few future directions that warrant investigation.

10.1 Future Work for Computational Efficiency

Which Technique to Use? A common question asked is how to decide which technique to use in practice? Unfortunately, there is no silver bullet given that we need to take the task, data, backbone, and hardware into consideration. Note that these techniques can often be combined for better results (to be discussed shortly).

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

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

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

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

10.2 Future Work for Data Efficiency

Advanced Sample Efficiency in Pretraining One of the most resource-intensive phases of language model development is pretraining, which traditionally requires vast amounts of data. Research into methods for improving sample efficiency—how effectively a model learns from a limited number of examples—could significantly reduce the data demands. Techniques such as self-supervised learning, which can extract more value from each data point by generating its own labels, show promise. Further investigation into novel self-supervised tasks and their impact on learning efficiency is needed.

Data Augmentation and Synthetic Data Enhancing the variety and volume of training data without manually collecting more samples is another avenue for improving data efficiency. Data augmentation techniques can artificially expand the training dataset, especially with existing proprietary model services. Moreover, using existing models as a filter for training data offers a way to create diverse and challenging datasets, potentially improving model generalization and robustness.

Few-shot and Zero-shot Learning The ability of a model to understand and perform tasks with very few examples (few-shot) or even without any task-specific data (zero-shot) represents a pinnacle of data efficiency. Exploring architectures and training methods that enhance a model's capacity for few-shot and zero-shot learning could lead to more versatile and broadly applicable models. More exploration into prompt learning and other types of data-efficient learning paradigm is still valuable.

Interactive and Incremental Learning Models that can learn interactively from user feedback or incrementally from new data streams can continuously improve while being deployed. This approach not only enhances data efficiency by learning from every interaction but also ensures that the model remains relevant and up-to-date with minimal manual intervention. Although approaches like reinforcement learning with human feedback can improve the performance of a language model with feedback from users, the best practice to periodically update the model remains unexplored.

This chapter, in part, are a reprint of the material as it appears in "A Survey on Model Compression and Acceleration for Pretrained Language Models" by Canwen Xu and Julian McAuley, which was published at *AAAI Conference on Artificial Intelligence*, 2023. The dissertation author was the primary investigator and author of this paper.

Bibliography

Daniel Adiwardana, Minh-Thang Luong, David R So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, et al. Towards a human-like open-domain chatbot. *arXiv preprint arXiv:2001.09977*, 2020.

Sungsoo Ahn, Shell Xu Hu, Andreas C. Damianou, Neil D. Lawrence, and Zhenwen Dai. Variational information distillation for knowledge transfer. In *CVPR*, 2019.

Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *NeurIPS*, 2016.

Stephen H. Bach, Victor Sanh, Zheng Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesht Sharma, Taewoon Kim, M. Saiful Bari, Thibault Févry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, Maged Saeed AlShaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, Xiangru Tang, Mike Tian-Jian Jiang, and Alexander M. Rush. Promptsource: An integrated development environment and repository for natural language prompts. In *ACL (Demos)*, 2022.

Jiangang Bai, Yujing Wang, Yiren Chen, Yaming Yang, Jing Bai, Jing Yu, and Yunhai Tong. Syntax-bert: Improving pre-trained transformers with syntax trees. In Paola Merlo, Jörg Tiedemann, and Reut Tsarfaty, editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 3011–3020. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.eacl-main.262. URL https://doi.org/10.18653/v1/2021.eacl-main.262.

Atilim Gunes Baydin, Robert Cornish, David Martínez-Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *ICLR*, 2018.

Asma Ben Abacha and Dina Demner-Fushman. A question-entailment approach to question answering. *BMC bioinformatics*, 20(1):1–23, 2019.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *FAccT*, 2021.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *EMNLP*, 2015.

Tom B. Brown, Benjamin Mann, Nick Ryder, et al. Language models are few-shot learners. In *NeurIPS*, 2020.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90% chatgpt quality. https://vicuna.lmsys.org/, 2023.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=r1xMH1BtvB.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Alexis Conneau and Douwe Kiela. Senteval: An evaluation toolkit for universal sentence representations. In *LREC*, 2018.

Jeffrey Cornelius-White. Learner-centered teacher-student relationships are effective: A meta-analysis. *Review of educational research*, 77(1):113–143, 2007.

Joe Davison, Joshua Feldman, and Alexander M. Rush. Commonsense knowledge mining from pretrained models. In *EMNLP-IJCNLP*, pages 1173–1178. Association for Computational Linguistics, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *IWP@IJCNLP*, 2005.

Mengnan Du, Subhabrata Mukherjee, Yu Cheng, et al. What do compressed large language models forget? robustness challenges in model compression. *arXiv preprint arXiv:2110.08419*, 2021.

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

Robert M Fano. Transmission of information: A statistical theory of communications. *American Journal of Physics*, 29(11):793–794, 1961.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, Proceedings of Machine Learning Research, 2017.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, September 2021a. URL https://doi.org/10.5281/zenodo.5371628.

Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. In *ACL-IJCNLP*. Association for Computational Linguistics, 2021b.

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

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

Demi Guo, Alexander M. Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. In *ACL-IJCNLP*, pages 4884–4896. Association for Computational Linguistics, 2021a.

Han Guo, Bowen Tan, Zhengzhong Liu, Eric P Xing, and Zhiting Hu. Text generation with efficient (soft) q-learning. *arXiv preprint arXiv:2106.07704*, 2021b.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International Conference on Machine Learning*, pages 3929–3938. PMLR, 2020.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: decoding-enhanced bert with disentangled attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=XPZIaotutsD.

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

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *ICLR*. OpenReview.net, 2021.

Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi. Knowledge transfer via distillation of activation boundaries formed by hidden neurons. In *AAAI*, 2019.

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

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *ICLR*. OpenReview.net, 2020.

Sara Hooker, Nyalleng Moorosi, Gregory Clark, Samy Bengio, and Emily Denton. Characterising bias in compressed models. *arXiv preprint arXiv:2010.03058*, 2020.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR, 2019.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *ICLR*. OpenReview.net, 2022.

Shengding Hu, Ning Ding, Huadong Wang, Zhiyuan Liu, Juanzi Li, and Maosong Sun. Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification. *arXiv preprint arXiv:2108.02035*, 2021.

Ting-Kuei Hu, Tianlong Chen, Haotao Wang, and Zhangyang Wang. Triple wins: Boosting accuracy, robustness and efficiency together by enabling input-adaptive inference. In *ICLR*, 2020.

Heinrich Jiang, Been Kim, Melody Guan, and Maya Gupta. To trust or not to trust a classifier. In *NeurIPS*, 2018.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. How can we know what language models know. *Trans. Assoc. Comput. Linguistics*, 8:423–438, 2020.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, et al. Tinybert: Distilling BERT for natural language understanding. In *EMNLP (Findings)*, 2020.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is BERT really robust? natural language attack on text classification and entailment. In *AAAI*, 2020.

Xiao Jin, Baoyun Peng, Yichao Wu, Yu Liu, Jiaheng Liu, Ding Liang, Junjie Yan, and Xiaolin Hu. Knowledge distillation via route constrained optimization. In *ICCV*, 2019.

Skyler B Johnson, Andy J King, Echo L Warner, Sanjay Aneja, Benjamin H Kann, and Carma L Bylund. Using chatgpt to evaluate cancer myths and misconceptions: artificial intelligence and cancer information. *JNCI Cancer Spectrum*, 7(2):pkad015, 2023.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *ICML*, 2019.

Pei Ke, Haozhe Ji, Siyang Liu, Xiaoyan Zhu, and Minlie Huang. Sentilare: Sentiment-aware language representation learning with linguistic knowledge. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6975–6988. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.567. URL https://doi.org/10.18653/v1/2020.emnlp-main.567.

Jack Kiefer, Jacob Wolfowitz, et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

Jangho Kim, Seonguk Park, and Nojun Kwak. Paraphrasing complex network: Network compression via factor transfer. In *NeurIPS*, 2018.

Taehyeon Kim, Jaehoon Oh, Nakyil Kim, Sangwook Cho, and Se-Young Yun. Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation. In *IJCAI*, pages 2628–2635. ijcai.org, 2021.

Young Jin Kim and Hany Hassan Awadalla. Fastformers: Highly efficient transformer models for natural language understanding. *arXiv preprint arXiv:2010.13382*, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *NeurIPS*, 1994.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *ICLR (Workshop)*, 2017.

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, et al. ALBERT: A lite BERT for self-supervised learning of language representations. In *ICLR*, 2020.

Anne Lauscher, Olga Majewska, Leonardo F. R. Ribeiro, Iryna Gurevych, Nikolai Rozanov, and Goran Glavas. Common sense or world knowledge? investigating adapter-based knowledge injection into pretrained transformers. *CoRR*, abs/2005.11787, 2020. URL https://arxiv.org/abs/2005.11787.

Teven Le Scao and Alexander M. Rush. How many data points is a prompt worth? In *NAACL-HLT*, pages 2627–2636. Association for Computational Linguistics, 2021.

Teven Le Scao, Angela Fan, Christopher Akiki, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Hector J. Levesque. The winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, 2011.

Yoav Levine, Barak Lenz, Or Dagan, Ori Ram, Dan Padnos, Or Sharir, Shai Shalev-Shwartz, Amnon Shashua, and Yoav Shoham. Sensebert: Driving some sense into BERT. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4656–4667. Association for Computational Linguistics, 2020. doi:

10.18653/v1/2020.acl-main.423. URL https://doi.org/10.18653/v1/2020.acl-main.423.

Yoav Levine, Barak Lenz, Opher Lieber, Omri Abend, Kevin Leyton-Brown, Moshe Tennenholtz, and Yoav Shoham. Pmi-masking: Principled masking of correlated spans. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=3Aoft6NWFej.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, et al. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-demo.21. URL https://aclanthology.org/2021.emnlp-demo.21.

Lei Li, Yankai Lin, Shuhuai Ren, Deli Chen, Xuancheng Ren, Peng Li, Jie Zhou, and Xu Sun. Accelerating pre-trained language models via calibrated cascade. *arXiv preprint arXiv:2012.14682*, 2020a.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL-IJCNLP*, pages 4582–4597. Association for Computational Linguistics, 2021.

Zekun Li, Wenhu Chen, Shiyang Li, Hong Wang, Jing Qian, and Xifeng Yan. Controllable dialogue simulation with in-context learning. *arXiv preprint arXiv:2210.04185*, 2022.

Zhuohan Li, Eric Wallace, Sheng Shen, et al. Train large, then compress: Rethinking model size for efficient training and inference of transformers. In *ICML*, 2020b.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *ACL*, pages 3214–3252. Association for Computational Linguistics, 2022.

Benlin Liu, Yongming Rao, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Metadistiller: Network self-boosting via meta-learned top-down distillation. In *ECCV*, 2020a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019a.

Weijie Liu, Peng Zhou, Zhiruo Wang, et al. Fastbert: a self-distilling BERT with adaptive inference time. In *ACL*, 2020b.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. K-BERT: enabling language representation with knowledge graph. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 2901–2908. AAAI Press, 2020c. URL https://ojs.aaai.org/index.php/AAAI/article/view/5681.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too. *arXiv preprint arXiv:2103.10385*, 2021.

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Avery Ma, Fartash Faghri, and Amir-massoud Farahmand. Adversarial robustness through regularization: A second-order approach. *arXiv preprint arXiv:2004.01832*, 2020.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *NeurIPS*, 2019.

Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *AAAI*, 2020.

Nelson Morgan and Hervé Bourlard. Generalization and parameter estimation in feedforward netws: Some experiments. In *NeurIPS*, 1989.

OpenAI. Chatgpt: Optimizing language models for dialogue, Jan 2023a. URL https://openai.com/blog/chatgpt/.

OpenAI. Gpt-4 technical report, 2023b.

Andrei Paleyes, Raoul-Gabriel Urma, and Neil D Lawrence. Challenges in deploying machine learning: a survey of case studies. *arXiv preprint arXiv:2011.09926*, 2020.

Haojie Pan, Chengyu Wang, Minghui Qiu, Yichang Zhang, Yaliang Li, and Jun Huang. Meta-kd: A meta knowledge distillation framework for language model compression across domains. *arXiv preprint arXiv:2012.01266*, 2020.

Tianyu Pang, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. Improving adversarial robustness via promoting ensemble diversity. In *ICML*, 2019.

Dae Young Park, Moon-Hyun Cha, Changwook Jeong, Daesin Kim, and Bohyung Han. Learning student-friendly teacher networks for knowledge distillation. *arXiv preprint arXiv:2102.07650*, 2021.

Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. Relational knowledge distillation. In *CVPR*, 2019.

Nikolaos Passalis and Anastasios Tefas. Learning deep representations with probabilistic knowledge transfer. In *ECCV*, 2018.

Kimessha Paupamah, Steven James, and Richard Klein. Quantisation and pruning for neural network compression and regularisation. *arXiv preprint arXiv:2001.04850*, 2020.

Baoyun Peng, Xiao Jin, Dongsheng Li, Shunfeng Zhou, Yichao Wu, Jiaheng Liu, Zhaoning Zhang, and Yu Liu. Correlation congruence for knowledge distillation. In *ICCV*, 2019.

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. True few-shot learning with language models. *arXiv preprint arXiv:2105.11447*, 2021.

Matthew E. Peters, Mark Neumann, Robert L. Logan IV, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. Knowledge enhanced contextual word representations. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 43–54. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1005. URL https://doi.org/10.18653/v1/D19-1005.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. Language models as knowledge bases? In *EMNLP-IJCNLP*, pages 2463–2473. Association for Computational Linguistics, 2019.

Hieu Pham, Zihang Dai, Qizhe Xie, Minh-Thang Luong, and Quoc V Le. Meta pseudo labels. *arXiv preprint arXiv:2003.10580*, 2020.

Hieu Pham, Xinyi Wang, Yiming Yang, and Graham Neubig. Meta back-translation. *arXiv preprint arXiv:2102.07847*, 2021.

Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

Guanghui Qin and Jason Eisner. Learning how to ask: Querying lms with mixtures of soft prompts. In *NAACL-HLT*, pages 5203–5212. Association for Computational Linguistics, 2021.

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

Colin Raffel, Noam Shazeer, Adam Roberts, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 2020.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*, 2016.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2015.

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

Victor Sanh, Thomas Wolf, and Alexander M. Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *NeurIPS*, 2020.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. Multitask prompted training enables zero-shot task generalization. In *ICLR*, 2022.

Timo Schick and Hinrich Schütze. Exploiting cloze-questions for few-shot text classification and natural language inference. In *EACL*, pages 255–269. Association for Computational Linguistics, 2021a.

Timo Schick and Hinrich Schütze. It's not just size that matters: Small language models are also few-shot learners. In *NAACL-HLT*, pages 2339–2352. Association for Computational Linguistics, 2021b.

Timo Schick, Helmut Schmid, and Hinrich Schütze. Automatically identifying words that can serve as labels for few-shot text classification. In *COLING*, pages 5569–5578. International Committee on Computational Linguistics, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Commun. ACM*, 63 (12):54–63, 2020a.

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. The right tool for the job: Matching model and instance complexities. In *ACL*, pages 6640–6651. Association for Computational Linguistics, 2020b.

Sheng Shen, Zhen Dong, Jiayu Ye, et al. Q-BERT: hessian based ultra low precision quantization of BERT. In *AAAI*, 2020.

Wenxian Shi, Yuxuan Song, Hao Zhou, Bohan Li, and Lei Li. Learning from deep model via exploring local targets, 2021. URL https://openreview.net/forum?id=5slGDu_bVc6.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *EMNLP*, pages 4222–4235. Association for Computational Linguistics, 2020.

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

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Naresh K. Sinha and Michael P. Griscik. A stochastic approximation method. *IEEE Trans. Syst. Man Cybern.*, 1(4):338–344, 1971.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

Samuel Stanton, Pavel Izmailov, Polina Kirichenko, et al. Does knowledge distillation really work? *arXiv preprint arXiv:2106.05945*, 2021.

Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4):585–591, 1997.

Thilo Strauss, Markus Hanselmann, Andrej Junginger, and Holger Ulmer. Ensemble methods as a defense to adversarial perturbations against deep neural networks. *arXiv preprint arXiv:1709.03423*, 2017.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *ACL*, pages 3645–3650. Association for Computational Linguistics, 2019.

Dong Su, Huan Zhang, Hongge Chen, Jinfeng Yi, Pin-Yu Chen, and Yupeng Gao. Is robustness the cost of accuracy? - A comprehensive study on the robustness of 18 deep image classification models. In *ECCV*, volume 11216 of *Lecture Notes in Computer Science*, pages 644–661. Springer, 2018.

Yusheng Su, Xu Han, Zhengyan Zhang, Yankai Lin, Peng Li, Zhiyuan Liu, Jie Zhou, and Maosong Sun. Cokebert: Contextual knowledge selection and embedding towards enhanced pre-trained language models. *AI Open*, 2:127–134, 2021. doi: 10.1016/j.aiopen.2021.06.004. URL https://doi.org/10.1016/j.aiopen.2021.06.004.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for BERT model compression. In *EMNLP-IJCNLP*, 2019a.

Tianxiang Sun, Yunfan Shao, Xipeng Qiu, Qipeng Guo, Yaru Hu, Xuanjing Huang, and Zheng Zhang. Colake: Contextualized language and knowledge embedding. In Donia Scott, Núria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 3660–3670. International Committee on Computational Linguistics, 2020a. doi: 10.18653/v1/2020.coling-main.327. URL https://doi.org/10.18653/v1/2020.coling-main.327.

Tianxiang Sun, Yunhua Zhou, Xiangyang Liu, et al. Early exiting with ensemble internal classifiers. *arXiv preprint arXiv:2105.13792*, 2021.

Yu Sun, Shuohuan Wang, Yu-Kun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. ERNIE: enhanced representation through knowledge integration. *CoRR*, abs/1904.09223, 2019b. URL http://arxiv.org/abs/1904.09223.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, et al. Mobilebert: a compact task-agnostic BERT for resource-limited devices. In *ACL*, 2020b.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, Proceedings of Machine Learning Research, 2019.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *ICPR*, 2016.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive representation distillation. In *ICLR*, 2020.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. Ensemble adversarial training: Attacks and defenses. In *ICLR*, 2018.

Frederick Tung and Greg Mori. Similarity-preserving knowledge distillation. In *ICCV*, 2019.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: The impact of student initialization on knowledge distillation. *arXiv preprint arXiv:1908.08962*, 13, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is all you need. In *NIPS*, 2017.

Elena Voita, David Talbot, Fedor Moiseev, et al. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *ACL*, 2019.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, et al. Superglue: A stickier benchmark for general-purpose language understanding systems. In *NeurIPS*, 2019a.

Alex Wang, Amanpreet Singh, Julian Michael, et al. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2019b.

Wenhui Wang, Furu Wei, Li Dong, et al. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *NeurIPS*, 2020.

Wenhui Wang, Hangbo Bao, Shaohan Huang, et al. Minilmv2: Multi-head self-attention relation distillation for compressing pretrained transformers. In *ACL-IJCNLP (Findings)*, 2021a.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. Kepler: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194, 2021b.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *ECCV*, 2018.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *TACL*, 2019.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *ICLR*. OpenReview.net, 2022.

Roman Werpachowski, András György, and Csaba Szepesvári. Detecting overfitting via adversarial examples. In *NeurIPS*, pages 7856–7866, 2019.

Alexander Wettig, Tianyu Gao, Zexuan Zhong, and Danqi Chen. Should you mask 15% in masked language modeling? *CoRR*, abs/2202.08005, 2022. URL https://arxiv.org/abs/2202.08005.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*, 2018.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi 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 M. Rush. Transformers: State-of-the-art natural language processing. In *EMNLP (Demos)*, pages 38–45. Association for Computational Linguistics, 2020.

Gloria Brown Wright. Student-centered learning in higher education. *International Journal of Teaching and Learning in Higher Education*, 23(1):92–97, 2011.

Hanwei Wu and Markus Flierl. Vector quantization-based regularization for autoencoders. In *AAAI*, pages 6380–6387. AAAI Press, 2020.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *CVPR*, 2016.

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

Ji Xin, Raphael Tang, Jaejun Lee, et al. Deebert: Dynamic early exiting for accelerating BERT inference. In *ACL*, 2020b.

Canwen Xu and Julian McAuley. A survey on model compression for natural language processing. *arXiv preprint arXiv:2202.07105*, 2022.

Canwen Xu, Wangchunshu Zhou, Tao Ge, et al. Bert-of-theseus: Compressing BERT by progressive module replacing. In *EMNLP*, 2020.

Canwen Xu, Wangchunshu Zhou, Tao Ge, et al. Beyond preserved accuracy: Evaluating loyalty and robustness of BERT compression. In *EMNLP*, 2021a.

Jingjing Xu, Wangchunshu Zhou, Zhiyi Fu, et al. A survey on green deep learning. *arXiv preprint arXiv:2111.05193*, 2021b.

Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. LUKE: Deep contextualized entity representations with entity-aware self-attention. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6442–6454, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.523. URL https://aclanthology.org/2020.emnlp-main.523.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, pages 5754–5764, 2019.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*, 2019.

Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *ICLR*, 2017.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *ACL*, pages 1–9. Association for Computational Linguistics, 2022.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *ACL*, pages 4791–4800. Association for Computational

Linguistics, 2019.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *NeurIPS*, 2015.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018a.

Ying Zhang, Tao Xiang, Timothy M. Hospedales, and Huchuan Lu. Deep mutual learning. In *CVPR*, 2018b.

Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. Dialogpt: Large-scale generative pre-training for conversational response generation. *arXiv preprint arXiv:1911.00536*, 2019a.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: enhanced language representation with informative entities. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 1441–1451. Association for Computational Linguistics, 2019b. doi: 10.18653/v1/p19-1139. URL https://doi.org/10.18653/v1/p19-1139.

Zexuan Zhong, Dan Friedman, and Danqi Chen. Factual probing is [MASK]: learning vs. learning to recall. In *NAACL-HLT*, pages 5017–5033. Association for Computational Linguistics, 2021.

Wangchunshu Zhou, Canwen Xu, Tao Ge, et al. BERT loses patience: Fast and robust inference with early exit. In *NeurIPS*, 2020.

Wangchunshu Zhou, Tao Ge, Ke Xu, and Furu Wei. Improving sequence-to-sequence pre-training via sequence span rewriting. *arXiv preprint arXiv:2101.00416*, 2021.

Wangchunshu Zhou, Canwen Xu, and Julian J. McAuley. BERT learns to teach: Knowledge distillation with meta learning. In *ACL*, 2022.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.