DSC250 Project Final Report: Efficient Adaption of Large Pre-Trained Models

James Chen
Halıcıoğlu Data Science Institute
La Jolla, CA 92093
mic033@ucsd.edu

Shuying Li
Halıcıoğlu Data Science Institute
La Jolla, CA 92093
shl146@ucsd.edu

Henry Wang
Halıcıoğlu Data Science Institute
La Jolla, CA 92093
shw103@ucsd.edu

Jiongli Zhu
Halıcıoğlu Data Science Institute
La Jolla, CA 92093
jiz143@ucsd.edu

Introduction

Recent applications of Large Language Models (LLMs), such as ChatGPT [1], have amplified the popularity of Pre-trained Language Models (PLMs) [2, 3], which serve as the foundation for these applications. By training on large datasets, PLMs learn universal language representations, which are useful for various downstream tasks such as text summarization and sentiment analysis. PLMs are usually fine-tuned to fit into specific application scenarios. However, previous research has found that directly fine-tuning the entire PLMs is often not efficient, requiring large amounts of data from downstream tasks or otherwise can suffer from overfitting [4, 5]. Therefore, efficient adaptation of those models has become a critical problem, which makes the application of PLMs in downstream tasks easier. There are three major types of methods for adapting PLMs efficiently:

- Fine-tuning only the selected parameters of the large models, thereby saving on computational resources and potentially mitigating overfitting [6, 7].
- Guiding the behavior of PLMs to achieve desired outcomes for specific tasks by crafting or learning specific prompts [8, 9, 10].
- Introducing additional layers (called adaptors) to be updated during fine-tuning and freezing the original model parameters, which ensures that the core knowledge embedded in the PLMs remains untouched while the adaptors tailor the model to the target task [11].

While each of these methods presents a unique perspective on efficient adaptation, a common thread running through them is the idea of reducing the computational cost via freezing most or all of the model parameters during fine-tuning. In this report, we present the results from experiments that (1) examine the efficiency and effectiveness of the vanilla approach, which directly fine-tunes/trains the base models on the data from the target domain, (2) compare the selected different kinds of adaptation approaches with the vanilla approach. We empirically compare these methods from the perspectives of both efficiency and effectiveness. Considering that the adaptation of PLMs is especially important in the case where we have limited data, we also present the results of experiments that examine the sensitivity of the adaptation methods to the training data size.

Literature Review

Generative pre-training (GPT) is a representative pre-trained and the first unidirectional language model, known for employing the fine-tuning approach to learn general language representations [12].

In response to the limitations of unidirectional techniques which contain limited left-to-right architecture, the Google AI team presented an advanced fine-tuning-based approach: Bidirectional Encoder Representations from Transformers (BERT). BERT employs a bidirectional training approach, using masked language modeling (MLM) across deep stacked layers to learn contextual information from both left and right sides of the input text [13]. During pre-training, BERT employs masked language modeling, randomly replacing some tokens with a "[MASK] token" and predicting the original token. The architecture design enables BERT to achieve State-of-The-Art (SOTA) performance on a variety of natural language processing tasks. As a result, the introduction of bidirectional language models has inspired further research in PTMs, such as Robustly Optimized BERT Pretraining Approach (RoBERTa) [14], decoding-enhanced BERT with disentangled attention (DeBERTa) [15], and Large-scale PTM GPT-3.

Due to the massive emergence of pre-trained language models with outstanding performance and the need for filling domain gaps with limited data from the target domain, effective adaptation methods have been a major focus of NLP-related research. However, fine-tuning the entire model is usually impractical because of the large number of parameters. Thus, most existing approaches seek to build fine-tuning frameworks that are less computationally demanding. One common method is adding one or more adaptation layers to the original model and freezing the pre-trained weights. For example, AdaMix [11] utilizes multiple mixed adaption modules in each transformation layer and freezes most of the pre-trained weights to create a parameter-efficient setup.

Partially freezing weights without additional modules is also an option. Normally, only a relatively small proportion of the layers are unfrozen for fine-tuning. Lee et al. [6] evaluated BERT [12] and RoBERTa [14] and concluded that about 25% of the final layers need to be tuned to retain 90% of the original performance. A relatively novel example of this type is BitFit [7] which concentrates on fine-tuning the bias parameters and successfully obtained decent performance.

Prompt tuning and prefix tuning are also popular for adapting language models. Prefix tuning [8] is a relatively novel approach that freezes LM parameters but only optimizes the continuous and task-specific prefix vector. Lester et al. [9] further proposed a simplified version of prefix tuning and illustrated that prompt tuning is a useful way to apply frozen models to downstream tasks and also enhances generalization for zero-shot domain adaptation. Another recent example is RLPROMPT [10], which seeks to mitigate the drawbacks of both soft prompts and discrete prompts by proposing an efficient method of discrete prompt optimization that adapts reinforcement learning to achieve a parameter-efficient prompt optimization framework.

Efficient Adaptation Methods for Pre-trained Language Models

In this section, we will discuss the adaptation of PLMs in detail. Let $M_{\theta^*_{src}}$ represent a PLM with parameters θ^*_{src}. These parameters θ^*_{src} are optimized to capture a broad understanding of natural language through extensive training on diverse text corpora and are designed to generalize across various linguistic tasks and domains. We can obtain the parameters θ^*_{src} by minimizing models' loss on the diverse source data D_{src}, i.e.,

$$\theta^*_{src} = argmin_\theta L(M_\theta, D_{src})$$

where L is the objective function that computes the loss based on the predictions of model M_θ. However, this generalization often comes at the cost of specialized performance in specific tasks. This will result in a high loss $L(M_{\theta^*_{src}}, D_{trg})$ when we directly apply the PLM $M_{\theta^*_{src}}$ to some given target domain, where D_{trg} is a dataset collected from the target domain. This necessitates the adaptation of $M_{\theta^*_{src}}$ to the target domain, which aims at modifying or fine-tuning the parameters original PLM θ^*_{src} into a new model θ^*_{trg} such that the adapted new model $M_{\theta^*_{src}}$ exhibits enhanced performance on the target domain. This objective can be formally defined as minimizing the loss function on the target domain:

$$\theta^*_{trg} = argmin_\theta L(M_\theta, D_{trg}).$$

However, this is different from directly training a model on the target domain from scratch in many ways. On the one hand, when adapting a PLM, we leverage the knowledge it has already acquired during its initial training phase on a large and diverse dataset D_{src}. This pretraining equips the model with a broad and general understanding of language. In contrast, training a model from scratch on the target domain only allows it to learn from the specificities of that domain, making the trained model more prone to overfitting. On the other hand, adaptation often requires significantly less data

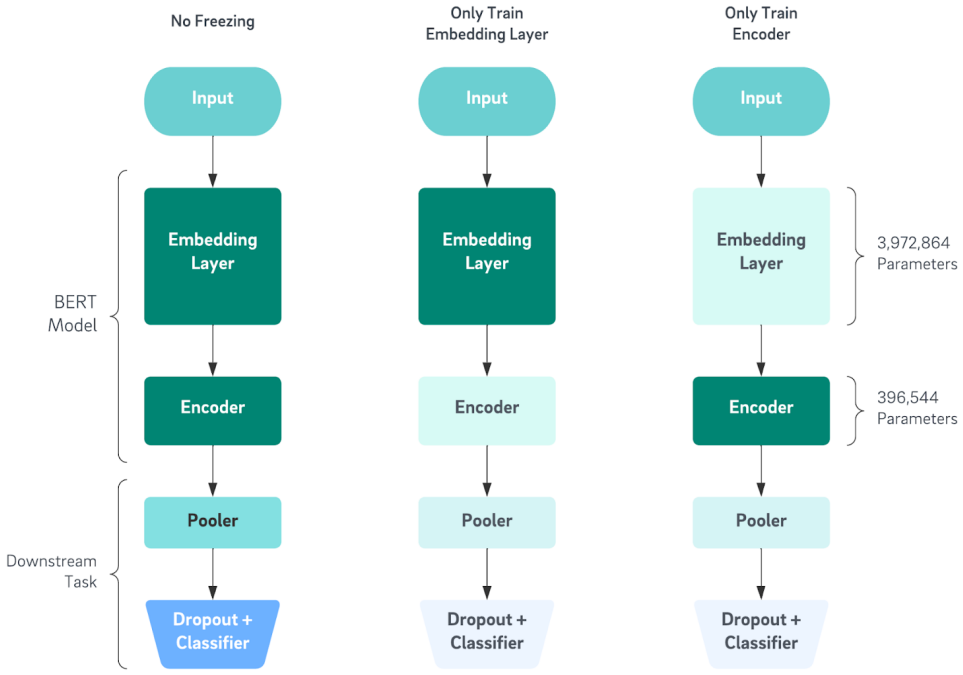

Figure 1: Diagram illustrating the naive approach we used for finetuning only the selected parameters.

and computational resources compared to training from scratch. Since the pretrained model has already developed a foundational language understanding, it only needs to adjust to the specifics of the target domain, which typically involves fine-tuning with a smaller, domain-specific dataset. This makes adaptation particularly valuable in scenarios where data availability is limited or where computational efficiency is a priority. Therefore, while the objective of minimizing the loss function on the target domain remains the same, the approach of adapting a pretrained model utilizes prior learning, requires fewer data and computational resources, and offers better generalization compared to training a model from scratch on the target domain.

In the following sections, we will discuss the main categories of efficient adaptation methods for PLMs.

Finetuning only the selected parameters

One efficient adaptation strategy is to fine-tune a selected subset of the model's parameters instead of the entire model. This approach, known as selective fine-tuning, involves identifying and updating only those parameters that are most critical for the performance on the target task. By doing so, computational resources are conserved, and the risk of overfitting is reduced. This method relies on the assumption that the pre-trained model has already learned a substantial amount of general knowledge, and only minimal updates are needed to tailor it to a specific task. For instance, one might only fine-tune the top layers of a transformer-based model like BERT, as these layers are often more task-specific compared to the lower layers which correspond to more general features.

Specifically, BitFit [7] is one of the most popular and widely used methods in this category. It is a sparse-finetuning approach that modifies only the bias terms of pre-trained BERT models. This method supports the hypothesis that fine-tuning is primarily about leveraging the knowledge induced by language-modeling training. By focusing on adjusting only the bias terms, BitFit provides an efficient way to fine-tune large language models while minimizing the number of parameters that need to be updated.

Introducing adaptors to be updated

Another method involves the insertion of small trainable modules, known as adaptors, into the pre-trained model architecture. These adaptors are designed to be lightweight and are trained while the rest of the model's parameters are kept frozen. This approach allows for the efficient transfer of knowledge from the PLM to the target task with a minimal increase in the number of parameters that need to be trained. Adaptors can take various forms, such as additional feed-forward layers or attention modules, and they can be inserted at different points within the model architecture.

Prompt tuning and prefix tuning

Prompt tuning and prefix tuning represent a paradigm shift in how PLMs are adapted. Instead of modifying the model itself, these techniques introduce changes to the input. Prompt tuning involves adding a prompt to the input text to guide the model towards the desired output. This prompt can be hand-crafted or learned from data. Prefix tuning is a related concept where a sequence of continuous vectors, referred to as a prefix, is prepended to the input embeddings. These vectors are optimized during training while the model parameters remain frozen. These approaches are especially useful for tasks where the input text can be naturally framed as completing a sentence or a paragraph, guiding the PLM to generate task-specific outputs. In this project, we adopt the classical method in this category during experiments [9]. This work proposes to add soft prompts to the embedding, which will be updated/trained during back-propagation. This approach is similar to the previous approach in the sense that the soft prompts can be seen as adaptors.

Experiments

Common Datasets and Evaluation Methods

There are various popular NLP downstream tasks for evaluating adaptation methods. For sentiment analysis tasks, experiments could be conducted on the IMDb dataset, which consists of movie reviews categorized as positive or negative [16]. For question-answering tasks, the Stanford Question Answering Dataset (SQuAD) offers a comprehensive collection of questions posed by crowd workers on a set of Wikipedia articles [17]. Named Entity Recognition (NER) experiments could benefit from the CONLL-2003 dataset, known for its reliability and extensive use in NER tasks [18]. Lastly, the AG News dataset which contains categorized news articles provides a solid foundation for text classification tasks [19]. In this report, due to the limited time, we will focus on the sentiment classification task on the IMDb dataset [16], which is a common choice in a range of related works [20, 21, 22].

Evaluation metrics are usually chosen based on the specific downstream tasks. For sentiment analysis and text classification, accuracy and F1-score are used as the primary metrics, which offer insights into the model's performance and ability to manage class imbalances [23]. In question-answering tasks, the Exact Match (EM) score and F1 score will be pivotal in assessing the precision of the model's responses against ground truths [17]. For NER, entity-level precision, recall, and F1-score will be employed to ensure a nuanced understanding of the model's entity identification capabilities [24]. Since we evaluated the IMDb data in this report, we will compare the accuracies for testing the effectiveness of fine-tuning. In addition to the effectiveness of the adaptation methods, the efficiency of adaptation in terms of computational resource requirements is also essential to the evaluation of adaptation methods, as this impacts the scalability and practical applicability of these methods in real-world scenarios. Therefore, we also tested the runtime and memory occupation of the methods in the experiments.

Our Experiment Settings

Due to the model sizes and time constraints, we adopt the TinyBERT and TinyRoBERTa models and are trying to analyze the performances of different fine-tuning/adaptation methods based on these models, including vanilla fine-tuning (all layers), selective fine-tuning (a subset of layers), prompt tuning, and BitFit. The IMDb dataset for binary sentiment classification is used for evaluation. We train each model for 10 epochs, and observe its runtime, training loss, and validation accuracy. By

default, 90% of the data are used for training and 10% are used for validation, and we also added experiments with a 30%/70% split to assess the approaches more in-depth.

Experimental Results

Epoch	1	2	3	4	5	6	7	8	9	10
Training Loss	0.56	0.43	0.40	0.37	0.35	0.34	0.32	0.32	0.31	0.30
Validation Accuracy	0.78	0.81	0.82	0.83	0.83	0.83	0.83	0.84	0.84	0.84
Runtime (s)	5	5	5	5	5	5	5	5	5	5

Table 1: TinyBERT with basic fine-tuning.

Epoch	1	2	3	4	5	6	7	8	9	10
Training Loss	0.67	0.60	0.51	0.46	0.43	0.41	0.40	0.39	0.39	0.39
Validation Accuracy	0.69	0.76	0.79	0.80	0.81	0.82	0.82	0.82	0.82	0.82
Runtime (s)	3	3	3	3	3	3	3	3	3	3

Table 2: TinyBERT with only fine-tuning the embedding layers.

Epoch	1	2	3	4	5	6	7	8	9	10
Training Loss	0.60	0.50	0.48	0.47	0.47	0.46	0.46	0.45	0.45	0.45
Validation Accuracy	0.76	0.78	0.79	0.79	0.79	0.80	0.80	0.80	0.80	0.80
Runtime (s)	4	4	4	4	4	4	4	4	4	4

Table 3: TinyBERT with only fine-tuning the encoder layers.

Epoch	1	2	3	4	5	6	7	8	9	10
Training Loss	0.61	0.53	0.51	0.50	0.49	0.49	0.48	0.48	0.48	0.47
Validation Accuracy	0.72	0.75	0.75	0.76	0.76	0.76	0.77	0.77	0.77	0.77
Runtime (s)	6	5	5	5	5	5	5	5	5	5

Table 4: TinyBERT with prompt tuning.

Epoch	1	2	3	4	5	6	7	8	9	10
Training Loss	0.69	0.69	0.68	0.68	0.68	0.68	0.67	0.67	0.67	0.67
Validation Accuracy	0.72	0.75	0.75	0.76	0.76	0.76	0.76	0.76	0.76	0.76
Runtime (s)	1	1	1	1	1	1	1	1	1	1

Table 5: TinyBERT with BitFit.

Epoch	1	2	3	4	5	6	7	8	9	10
Training Loss	0.58	0.37	0.31	0.28	0.25	0.23	0.21	0.19	0.18	0.17
Validation Accuracy	0.77	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.83
Runtime (s)	13	12	12	12	12	12	12	12	12	12

Table 6: TinyRoBERTa with basic fine-tuning.

Epoch	1	2	3	4	5	6	7	8	9	10
Training Loss	0.69	0.68	0.65	0.61	0.56	0.52	0.50	0.47	0.46	0.46
Validation Accuracy	0.71	0.72	0.73	0.75	0.76	0.77	0.78	0.79	0.80	0.80
Runtime (s)	9	9	9	9	9	9	9	9	9	9

Table 7: TinyRoBERTa with only fine-tuning the embedding layers.

Epoch	1	2	3	4	5	6	7	8	9	10
Training Loss	0.69	0.62	0.58	0.57	0.56	0.55	0.54	0.53	0.53	0.52
Validation Accuracy	0.62	0.69	0.69	0.70	0.72	0.72	0.73	0.73	0.73	0.74
Runtime (s)	11	10	10	10	10	10	10	10	10	10

Table 8: TinyRoBERTa with only fine-tuning the encoder layers.

Epoch	1	2	3	4	5	6	7	8	9	10
Training Loss	0.69	0.64	0.61	0.60	0.59	0.58	0.57	0.56	0.55	0.55
Validation Accuracy	0.57	0.65	0.67	0.68	0.68	0.69	0.70	0.70	0.70	0.70
Runtime (s)	12	12	12	12	12	12	12	12	12	12

Table 9: TinyRoBERTa with prompt tuning.

Epoch	1	2	3	4	5	6	7	8	9	10
Training Loss	0.55	0.50	0.47	0.45	0.44	0.43	0.43	0.42	0.42	0.42
Validation Accuracy	0.78	0.80	0.81	0.81	0.81	0.82	0.82	0.82	0.82	0.82
Runtime (s)	2	2	2	2	2	2	2	2	2	2

Table 10: TinyRoBERTa with BitFit.

Sensitivity to Training Data Size

Another critical issue in model adaptation is how data size affects performance. To evaluate this, we experimented with only 30% of training data. The results are listed in Tables 11 & 12.

	All	Embedding	Encoder	Prompt	BitFit
90%/10%	0.84	0.82	0.80	0.77	0.76
30%/70%	0.80	0.74	0.78	0.75	0.65

Table 11: Test accuracies varying training data size, TinyBERT.

	All	Embedding	Encoder	Prompt	BitFit
90%/10%	0.83	0.80	0.74	0.70	0.82
30%/70%	0.82	0.73	0.69	0.66	0.77

Table 12: Test accuracies varying training data size, TinyRoBERTa.

We managed to obtain the number of trainable parameters for some of the approaches. For TinyBERT, the numbers of parameters in the full network, the embedding layers, and the encoder layers are respectively 4,385,920, 3,972,864, and 396,544. The corresponding numbers in TinyRoBERTa are 16,979,776, 12,999,936, and 3,979,840. TinyRoBERTa would demand significantly more GPU memory than TinyBERT.

Analyses of the Results

Training loss: The training loss keeps decreasing for all methods. However, it is noteworthy that although TinyRoBERTa with basic fine-tuning (Table 6) has a much lower training loss than the others, its accuracy eventually falls within the same range as BitFit (Table 10). This observation can be due to overfitting. For the other adaptation methods, the training loss positively correlates with the validation accuracy, which is reasonable and indicates less overfitting.

Accuracy: Overall, the final accuracies of the BitFit methods are better than others (except for direct fine-tuning). Among other adaptation methods, prompt tuning with token length=20 yields the lowest accuracies, which might be improved with a larger length of soft prompts.

Convergence: Among all approaches, TinyRoBERTa with direct fine-tuning (Table 6) converges the fastest and reaches the converged accuracy in epoch 2. However, the method in Table 7 (TinyRoBERTa with selective fine-tuning) converges the most slowly as its accuracy increases at a visibly low rate. Besides, freezing the bottom layers generally leads to a lower starting accuracy and consequently a slower convergence.

Runtime: The runtime is affected by both the base model and the fine-tuning approach. Regardless of the adaptation method, TinyRoBERTa requires a significantly longer training time in each epoch (2 to 3 times that of TinyBERT). With the same base model, freezing the bottom layers decreases the training time since less computation is required for parameter updates. BitFit has the least runtime for both base models as it requires the fewest number of parameters.

Overall best: We consider the second approach (freezing the bottom layers for TinyBert) and the last one (TinyRoBERTa with BitFit) as the overall best models. This is because they are among the best in terms of accuracy while requiring less runtime as well as fewer trainable parameters. TinyRoBERTa with BitFit is the better of the two since it is even faster and is less sensitive while varying training data sizes.

Limitations

Since our work is analytical, we look at the respective limitations of different methods. For example, vanilla fine-tuning has the best accuracy but is demanding in computational resources. Fine-tuning the embedding layers is limited in its accuracy with fewer training data. It can be seen that with TinyBERT, fine-tuning the encoder layers actually overtakes the embedding layers while the training data is scarce. While prompt tuning appears to be outperformed by the other methods, which is its limitation, it does not lose much accuracy with 30% of training data. As for BitFit, the performance may be affected by the base model it is applied to, i.e., it is dependent on the base model.

Conclusions

We have tested the performances of the TinyBERT and TinyRoBERTa models as well as multiple fine-tuning methods for both of them. Generally speaking, the vanilla fine-tuning performs the best for both base models. Among the more efficient approaches, selectively fine-tuning the embedding layers and BitFit are relatively the better ones, as the former exhibits approximately optimal accuracy with highly improved runtime and memory occupation, and the latter works especially well with TinyRoBERTa.

In real-life use cases, one may need to carefully select which method to adopt since each one has its pros and cons. For instance, BitFit for TinyRoBERTa would be the optimal choice when there is a tight constraint on computational resources. When such a constraint does not exist, however, the vanilla (full network) fine-tuning may be the most reasonable option.

References

[1] OpenAI. Chatgpt. `https://www.openai.com/chatgpt/`, 2023. Computer software.

[2] Sergey Edunov, Alexei Baevski, and Michael Auli. Pre-trained language model representations for language generation. *arXiv preprint arXiv:1903.09722*, 2019.

[3] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*, 56(2):1–40, 2023.

[4] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, 2022.

[5] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*, 2020.

[6] Jaejun Lee, Raphael Tang, and Jimmy Lin. What would elsa do? freezing layers during transformer fine-tuning. *arXiv preprint arXiv:1911.03090*, 2019.

[7] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

[8] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

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

[10] Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. *arXiv preprint arXiv:2205.12548*, 2022.

[11] Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. Adamix: Mixture-of-adapter for parameter-efficient tuning of large language models. *arXiv preprint arXiv:2205.12410*, 1(2):4, 2022.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[13] Haifeng Wang, Jiwei Li, Hua Wu, Eduard Hovy, and Yu Sun. Pre-trained language models and their applications. *Engineering*, 25:51–65, 2023.

[14] 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, 2019.

[15] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention, 2021.

[16] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, 2011.

[17] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[18] Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2003*, pages 142–147, 2003.

[19] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657, 2015.

[20] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[21] 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.

[22] Maël Fabien, Esaú Villatoro-Tello, Petr Motlicek, and Shantipriya Parida. Bertaa: Bert fine-tuning for authorship attribution. In *Proceedings of the 17th International Conference on Natural Language Processing (ICON)*, pages 127–137, 2020.

[23] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[24] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.