FAST: Feed-Forward Assisted Transformers for Time Efficient Fine Tuning

Anonymous ACL submission

Abstract

Fine-tuning is the standard approach when adapting pre-trained large language models for specific downstream tasks. However, the energy and time required to fully fine-tune all parameters can become prohibitively large for many applications as the size of the model increases. While recent advancements in parameter-efficient transfer learning have reduced the number of parameters that need to be updated, the training time and energy consumption of these methods remain similar to full fine-tuning. In this paper, we propose a time-efficient fine-tuning method based on feature-extraction in which we treat off-the-shelf language models as fixed sources of embeddings and train small feed-forward networks on top for each downstream task. Averaging across the GLUE NLI benchmark, our method trains 124 times faster than full fine-tuning and 101 times faster than parameter-efficient fine-tuning methods using *distilRoBERTa*[1], while achieving 81.9% and 85.0% performance respectively.

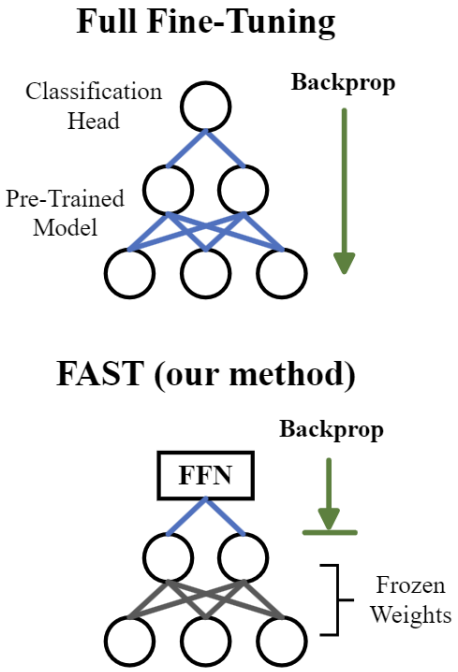

Figure 1: Visualization of FAST and comparison to standard fine-tuning.

1 Introduction

Enhancing computational efficiency stands as a significant challenge within deep learning (Rolnick et al., 2019; Strubell et al., 2019; Min et al., 2021). The issue of efficiency is particularly prevalent in the subfield of natural language processing (NLP), where it is common practice to fully fine-tune pre-trained large language models (LLMs) for specific downstream tasks (Hendrycks et al., 2019; Andreassen et al., 2021; Miller et al., 2021), where full fine-tuning refers to fine-tuning all model parameters. However, full fine-tuning of LLMs becomes prohibitively costly as models become increasingly complex (Ding et al., 2023).

Parameter-efficient fine-tuning (PEFT) methods aim to address this by fine-tuning only a small subset of model parameters and freezing the rest (Ding et al., 2023; Mudrakarta et al., 2018). Common PEFT methods include adapters (Houlsby et al., 2019), prefix tuning (Li and Liang, 2021) and soft prompt tuning (Lester et al., 2021).[2] Efficiency for PEFT methods is generally quantified by a reduction in the number of parameters used (Howard et al., 2017; Sandler et al., 2018; Henderson et al., 2020). In practice, however, training times are also an important metric to consider for efficiency (Cao et al., 2020) – lengthy training times and substantial energy consumption commonly hinder progress in the field, and the growing computational burden of deep learning is projected to become prohibitive in many academic and industry applications (Thompson et al., 2020). Fu et al. (2022) demonstrated that commonly used PEFT methods incur longer

[1] https://huggingface.co/distilbert/distilroberta-base

[2] Discussed in more detail in §2

training times than full fine-tuning, despite using fewer parameters.

A simple approach to avoid the prohibitive time cost of PEFT is to "freeze" the weights of the pre-trained model and train a separate network on top of the model's embeddings (Min et al., 2021). This is reminiscent of classic NLP feature-based techniques (Koehn et al., 2003). In particular, the approach involves choosing the architecture of the additional network and the specific model embeddings. The simplest approach fine-tunes a single layer classification head on the last layer embedding of the transformer, commonly referred to as linear probing (Kumar et al., 2022).

In this work, we introduce FAST (Feed-Forward ASsisted Transformer), which maintains the efficiency of feature-based fine-tuning, but enhances performance by utilizing more recent embedding methods and stacking layers of MLP on the outputs of a pre-trained language model. We present the following core contributions:

- An efficiency focused analysis of FAST, which represents a 80-170x speedup compared to full and common parameter-efficient tuning methods
- A comprehensive view of embedding extraction from pre-trained transformers, including sentence transformers
- Comparison of different concatenation methods for transformer-based embeddings on multi-sentence tasks
- Analysis of intrinsic dimensionality of various embedding methods and their relation to classification ability

2 Related Work

Adapters In adapter fine-tuning, small feed-forward network modules called *adapters* are inserted between transformer layers (Houlsby et al., 2019). The network modules include fully-connected down and up projections in each layer, where an input feature $h \in \mathbb{R}^d$ is mapped to a lower r-dimensional space using a transformation matrix $W_d \in \mathbb{R}^{d \times r}$. Subsequently, a nonlinear activation function $f(\cdot)$ is applied, and the feature is mapped back to the original d-dimensional space using $W_r \in \mathbb{R}^{r \times d}$. A residual connection then adds the original input feature to obtain the final output:

$$h' = f(h * W_d) * W_r + h \quad (1)$$

Prompt Tuning Prompt tuning is a method that incorporates task-specific prompts into frozen input embeddings during fine-tuning . There are two prominent categories within prompt tuning: prefix fine-tuning and soft prompt tuning. In prefix fine-tuning, a set of trainable tokens is prepended to the inputs and hidden states of each transformer layer (Li and Liang, 2021). Soft prompt tuning simply appends a trainable prompt to the original input of the language model (Lester et al., 2021).

Feature Extraction Peters et al. (2019) compares full fine-tuning and feature extraction methods. They propose the use of more complex architectures on top of frozen pre-trained models, varying for different types of NLP tasks. These include bi-attentive classification networks and BiLSTMs on a linear combination of multiple different hidden states from the pre-trained model. Wang et al. (2019) propose the use of both fine-tuning and feature extraction, training the weights of the network and an additional LSTM to further increase benchmark scores. However, both primarily focus on analyzing accuracy without comparing the efficiency of their proposed methods.

3 Method

This section focuses on the two main aspects of our model: the feed-forward network and the method of generating embeddings from the base pre-trained language model.

3.1 Feed-Forward Network

We perform feature extraction on pre-trained large language model with frozen weights by extracting the embeddings from the last layer. We discuss methods for generating the embeddings in §3.2 and §3.3. These embeddings are passed through a feed forward network, where the last layer of the feed-forward network is the classification layer.

For n layers and input embeddings $X \in \mathbb{R}^{N \times d}$ with N tokens and dimension d, the hidden states and output of the FFN are denoted as

$$\begin{aligned} H_1 &= \text{ReLU}(X * W_1 + B_1) \\ H_2 &= \text{ReLU}(H_1 * W_2 + B_2) \\ &\vdots \\ H_i &= \text{ReLU}(H_{i-1} * W_i + B_i) \\ &\vdots \\ Y &= \text{ReLU}(H_{n-1} * W_{n-1} + B_{n-1}) \end{aligned}$$

where $W_i \in \mathbb{R}^{d \times d}$ is a learnable matrix, and $B_i \in \mathbb{R}^d$ is the learnable bias. During implementation, the number of feed-forward layers is obtained through a grid search of the set $\{1, 3, 5\}$.

3.2 Embeddings

We explore three different techniques for extracting a fixed-length embedding from the output of the language model for a given input sequence: CLS token embedding, mean pooling, and sentence transformers.

CLS While RoBERTa is not pre-trained to have a semantically meaningful representation for the [CLS] token (Liu et al., 2019), we do use the [CLS] token as an embedding option to act as a baseline.

Mean Pool To represent the average semantic meaning of a given input sentence, We extract the output embeddings, excluding padding, from the language model and compute the element-wise mean (Chen et al., 2018).

Sentence Transformer Sentence transformers (Reimers and Gurevych, 2019) uses siamese network structures to obtain sentence embeddings that can be compared with cosine similarity. Sentence transformers are useful for multi-sentence tasks and are robust to zero-shot learning (Biesner et al., 2022). We use a sentence transformer for multi-sentence tasks and directly use the output embedding as the input to the feed-forward network.

3.3 Concatenation

For tasks with multiple input sentences, we need to generate a joint embedding to represent the entire input to pass into the feed-forward network. First, we generate embeddings for each sentence individually, denoted U and V, using one of the aforementioned methods. The way these individual embeddings are concatenated can significantly affect the performance (Reimers and Gurevych, 2019). In our approach, we use either $|U - V|$ or $U - V$ as the input to the FFN, depending on whether the order of input sentences for the particular task is meaningful.[3]

3.4 Benchmarking

We benchmarked performance with respect to time and energy consumption for the feed-forward network. We use real time power estimation for energy consumption (García-Martín et al., 2019).

For a given training, we recorded the time spent on each epoch of the training loop and compute the average across all epochs. We obtain our estimate for the overall training time as the average training time per epoch multiplied by the number of epochs that passed during training. The energy consumption in joules (J) from the estimated training times is

$$E = \int_t P dt \tag{2}$$

where P is the real-time power consumption output of the device in watts (W) and t is the time in seconds. The value of P is obtained directly from the user's hardware.

4 Experiments and Results

Section 4.1 introduces the specific model and datasets used in our experiments. Section 4.2 reports our experimental results for accuracy. Section 4.3 reports our experimental results for computational efficiency and energy efficiency comparisons to other leading delta-tuning measures. Section 4.4 compares the performance of different concatenation methods on multi-sentence tasks. Section 4.5 compares how our method's performance generalizes to other pre-trained models. Section 4.6 utilizes manifold learning techniques to analyze different embedding techniques, and help explain our results.

4.1 Experimental Setup

Dataset and Evaluation Metrics We evaluated the performance of our method across 10 tasks sourced from the General Language Understanding Evaluation (GLUE) dataset[4]. This dataset comprises a diverse set of linguistic tasks used to assess the generalization and proficiency of natural language understanding models (Wang et al., 2018b). To evaluate computational efficiency, we report the estimated time cost and hardware energy consumption of our experiments.

Implementation We chose distilRoBERTa, a distilled version of the RoBERTa model, as our base model to generate embeddings. It follows the same training procedure as distilBERT, as outlined in the associated paper (Sanh et al., 2019). The model's architecture consists of 6 transformer layers, each

[3] For example, the QQP task in the GLUE benchmark tests if two questions are semantically identical, so the order the sentences are given to the model should not change the prediction.

[4] See Appendix B

	GLUE Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI_m	QNLI	RTE	WNLI
CLS	60.9	35.3	**87.1**	**78.6**	31.3	52.9	66.3	71.2	50.3	**65.1**
Mean Pool	**64.7**	**38.9**	86.9	70.6	66.1	**59.3**	**67.3**	**74.1**	50.3	**65.1**
ST	63.4	15.9	84.8	76.0	**80.6**	54.7	66.8	71.9	**55.9**	58.2
DistilRoBERTa	83.2	59.3	92.5	86.6	88.3	89.4	84.0	90.8	67.9	-
BERT Adapter	80.2	59.2	94.3	84.3	86.1	89.4	85.4	92.4	71.6	65.1
CBoW	58.6	0.0	80.0	73.4	61.2	51.4	56.0	72.1	54.1	65.1

Table 1: GLUE Benchmark scores (MCC, F1, accuracy) of the CLS, Mean Pool, and Sentence Transformer methods, and comparisons with other methods. CoLA is evaluated using MCC. MRPC and QQP are evaluated using F1 score. STS-B uses Spearman's correlation coefficient. The other tasks use accuracy.

with 12 heads and an embedding dimension of 768. We selected this base model for its accessibility, strong performance on NLP tasks, cost-effectiveness in terms of computational resources, and the availability of a publicly accessible pre-trained sentence transformer.

The weights of distilRoBERTa remain frozen during tuning, allowing us to precompute and store the model's output embeddings for all sentences in the dataset. This initial computation serves as a one-time cost before training our additional parameters directly. The saved embeddings are used during training of the feed-forward network, eliminating the need to run inference on the language model in each epoch. To optimize the FFN's performance, we perform a hyperparameter grid search to determine the best values for number of layers (1, 3, or 5), batch size, learning rate, and weight decay[5]. All computation was performed on one V100 GPU.

4.2 GLUE Benchmark Results

We assessed our method's performance against both state-of-the-art models and baseline techniques that undergo training within a similar computational time frame. Specifically, we compare against a full fine-tune of DistilRoBERTa, adapters, and CBoW. The performance outcomes across GLUE tasks are reported in Table 1.

Overall, we observed that mean pooling tends to yield the best performance among the three types of embeddings. This is likely because the pre-trained model is optimized to generate meaningful embeddings for predicting the next token. Since mean pooling incorporates information from all tokens in the sequence, it may also provide a more consistent representation of the entire text regardless of the downstream task. On the other hand, RoBERTa's start [CLS] token is not pretrained on any task, so distilRoBERTa likely lacks a meaningful [CLS] token.

[5]See Appendix A

4.3 Energy Efficiency

In Table 2, we evaluated time cost and energy consumption across GLUE tasks for our method, as well as for DistilRoBERTa and CBoW, according to the method described in Section 3.4. Note that generating the embeddings of the language model is not included in the training time, though this is a one-time cost that is amortized across training runs. We reported these efficiency metrics for 1, 3 and 5 layers of our FFN. FAST demonstrates a 124x training speedup over fully fine-tuning all model weights of a distilRoBERTa transformer model. FAST also demonstrated a 101x speedup over common PEFT methods. Due to this reduced training time, the energy consumption of FAST is also 124x less than full fine-tuning and 101x less than PEFT methods.

Figure 2 illustrates that although FAST and common PEFT methods utilize similar parameter counts, training for FAST is two orders of magnitude faster than that of PEFT methods. Furthermore, both FAST and PEFT outperform full fine-tuning in terms of parameter count and training time. This demonstrates that FAST provides greater time efficiency at comparable parameter counts to common PEFT methods.

4.4 Concatenation Method

For multi-sentence tasks, we experimented with various methods of concatenating sentence embeddings. Previous work has shown significant impact from different aggregations of sentence embeddings on performance (Reimers and Gurevych, 2019). Table 3 presents a comparative analysis of different concatenation methods (as before, U and

	Time (s)			Energy (J)		
	CoLA	**MRPC**	**QNLI**	**CoLA**	**MRPC**	**QNLI**
1-layer FAST	0.72	0.31	7.96	181	76	1990
3-layer FAST	0.98	0.45	12.09	246	111	3023
5-layer FAST	1.29	0.57	15.93	323	143	3983
Prompt Tuning	92.08	39.72	1125.00	23020	9930	281250
Prefix Tuning	92.72	40.10	1136.00	23180	10025	284000
Adapter	101.53	43.67	1221.80	25383	10918	305450
Full Fine-Tuning	116.45	50.13	1408.00	29113	12534	352000

Table 2: Training times (s) and energy consumption (J) across GLUE tasks for the CLS, Mean Pool, and Sentence Transformer methods on a V100 GPU, and comparison with full fine-tuning.

		MRPC	**STS-B**	**QQP**	**QNLI**	**RTE**	**WNLI**
Mean Pooling	U, V	72.55	33.57	74.98	68.00	56.68	56.34
	$\|U - V\|$	80.15	78.89	75.87	73.16	59.57	56.34
	$U, V, \|U - V\|$	72.55	67.35	76.91	70.90	57.04	56.34
CLS	U, V	71.08	26.80	73.15	62.75	56.68	56.34
	$\|U - V\|$	79.41	74.57	76.38	70.95	58.12	56.34
	$U, V, \|U - V\|$	71.32	71.35	76.92	67.16	56.34	56.34

Table 3: Comparison of different concatenation methods using distilRoBERTa across multi-sentence GLUE tasks, using both CLS and Mean Pooled embeddings.

V denote the two individual sentence embeddings) for different multi-sentence GLUE tasks. Employing $U - V$ notably enhanced performance, and so was used in our overall performance benchmark comparing to full fine-tuning.

4.5 Generalization to other Pre-Trained Models

To ensure that our method generalizes to models beyond just distilRoBERTa, we ran experiments using our same method on a different public transformer: MPNet (Song et al., 2020). Table 4 shows the results across 8 GLUE benchmark tasks, where we observe similar patterns to our method's performance with distilRoBERTa. Mean pooling and sentence transformers consistently outperform CLS, with sentence transformers being particularly effective at regression tasks like STS-B, but poor at grammar tasks like CoLA.

4.6 Dimensionality Analysis

We use dimensional analysis and manifold learning techniques to establish a quantifiable basis for comparing different embedding models (distilRoBERTaST, distilRoBERTaBase (mean pooled), and CBoW), as well as a way to conceptually explain the efficacy of the FFN.

Previous work from Tulchinskii et al. (2023) and Mamou et al. (2020) also suggests that of human language is intrinsically represented by a small number of dimensions. We posit that a lower intrinsic dimensionality may indicate a more straightforward task for the FFN insofar as the embeddings efficiently encapsulate the essential information required for the model to generalize effectively (Aghajanyan et al., 2020).

Note that we do not necessarily directly correlate lower dimensionality with enhanced performance, as the actual performance of FAST varies based on several other task-dependent and FFN-dependent parameters.

PCA Analysis: The embeddings are processed through Randomized PCA and the cumulative variance ratio is calculated to determine the RandPCA intrinsic dimensionality of the embedding, i.e. the minimum percentage of components required to surpass the 90% cumulative variance threshold.

Our results (Table 5) indicate that mean pooling and sentence transformers consistently generate embeddings that have lower intrinsic dimensionality than cBOW, suggesting that such embeddings are more effective at encoding critical information.

t-SNE Analysis: The t-SNE visualization provides an intuitive understanding of the data's seg-

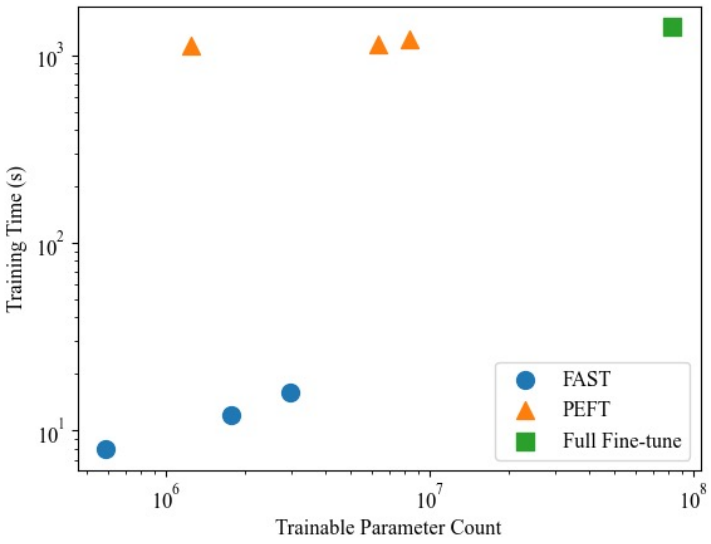

Figure 2: Log scale plot of training time (seconds) vs. trainable parameter count on the QNLI task for FAST (using MPNetBase and MPNetST), three common PEFT methods (adapters, prefix tuning, and soft prompt tuning), and full fine-tuning.

	CoLA	SST-2	MRPC	STS-B	QQP	QNLI	RTE	WNLI
MPNetBase (CLS)	41.16	85.44	71.08	65.73	75.31	65.84	56.32	56.34
MPNetBase (Meanpool)	**45.21**	89.45	**79.90**	75.96	77.05	71.48	62.46	56.34
MPNetSentence	27.70	**90.02**	77.21	**83.53**	**87.92**	**73.73**	**68.23**	**60.56**

Table 4: GLUE benchmark scores using FAST with embeddings from MPNet Base Transformer and MPNet Sentence Transformer.

mentation and separation (van der Maaten and Hinton, 2008). We observe the reduction in the intrinsic dimensionality of the embeddings (or lack thereof) as they move forward through the FFN. Our analyses (specific examples using the SST2 task embeddings are in Fig. 3) indicate that our simple FFN is sufficiently parameterized to dramatically reduce the intrinsic dimensionality of the embeddings generated from both mean pooling and sentence transformers, but that it struggles to unravel the underlying structure of the CBoW-generated embeddings. Our results further support the hypothesis that the intrinsic dimensionality from the last hidden layer of the FFN is a strong indicator of the classification and generalization ability of the model (Ansuini et al., 2019). This is consistent with the superior performance of mean-pooled embeddings and sentence transformer embeddings in testing.

5 Discussion

Our results suggest that our proposed method has promising implications for improving the accessibility of fine-tuning LLMs, particularly for those with limited computational resources. Most notably, our method offers substantially improved training efficiency, while maintaining relatively high levels of performance. Our method is notably compatible with inexpensive or resource-constrained hardware, which could empower individuals without access to high-end computational resources to engage in meaningful natural language processing tasks.

In this paper, we propose FAST (Feed-Forward Assisted Transformer), a novel addition-based delta tuning method that trains small feed forward neural networks for specific downstream tasks on top of embeddings generated from pre-trained off-the-shelf large language models. Our approach achieves up to 93.9% accuracy of full fine-tuning and accomplishes downstream tasks up to 170 times faster. Our method significantly reduces training time compared to alternative methods in parameter-efficient fine tuning, enabling more accessible natural language processing for users with limited hardware capabilities.

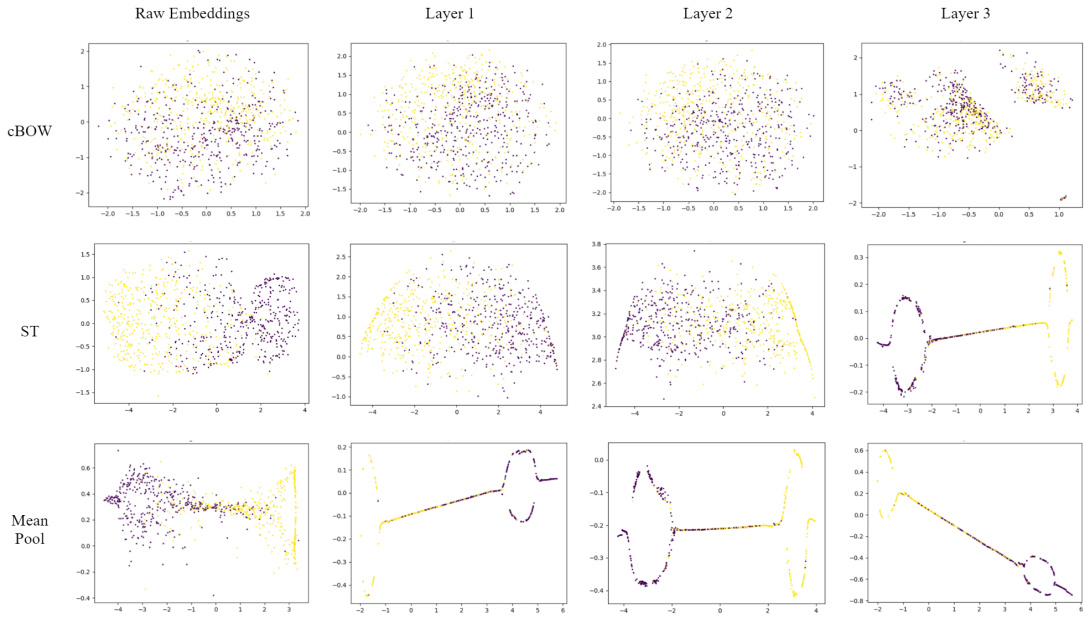

Figure 3: t-SNE plots of validation set SST2 embeddings at incremental layers of the trained FFN with 3 hidden layers.

	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE
Mean Pool	**21.22**	**16.67**	**54.30**	**55.34**	**52.60**	**50.52**	**28.26**	**23.44**
ST	28.91	34.11	66.67	69.79	64.58	73.57	37.37	34.51
CBoW	56.00	62.00	80.00	81.00	78.00	79.00	66.00	65.00

Table 5: RandPCA intrinsic dimensionality of the Mean Pool, Sentence Transformer, and Continuous Bag of Words embeddings.

Limitations

When interpreting our findings, there are a number of experimental limitations that should be considered. Firstly, our experiments were conducted on a V100 GPU. To gain a more comprehensive understanding of the performance and efficiency of our method, it would be beneficial to conduct tests on various machines and assess potential variations across different hardware configurations. Although the GLUE benchmark is a widely used standard in NLP research, it is ultimately limited to short, English language text inference tasks. Exploring tasks that involve larger levels of text granularity (e.g. paragraphs), as well as incorporating non-English language data, could provide valuable insights and enhance the generalizability of our method to a broader spectrum of linguistic challenges. We could also test our model against the SuperGLUE benchmark to assess performance on more challenging language tasks compared to the GLUE benchmark (Sarlin et al., 2019), which may yield more robust and comprehensive results. In addition, future research could explore additional types of embeddings and embedding concatenation strategies to assess their impact on model performance, as different kinds of inputs to the FFN may capture distinct linguistic features. Finally, we utilized time and energy consumption during training as our measure of efficiency. However, investigating other efficiency metrics, such as FLOP counts or memory consumption (Schwartz et al., 2019; Henderson et al., 2020), could yield further insights into the trade-offs between model performance and computational cost.

We do not foresee any unique risks that are not present in any research on large language models.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *CoRR*, abs/2012.13255.

Anders Andreassen, Yasaman Bahri, Behnam Neyshabur, and Rebecca Roelofs. 2021. The evo-

lution of out-of-distribution robustness throughout fine-tuning. *CoRR*, abs/2106.15831.

Alessio Ansuini, Alessandro Laio, Jakob H. Macke, and Davide Zoccolan. 2019. Intrinsic dimension of data representations in deep neural networks. *CoRR*, abs/1905.12784.

David Biesner, Maren Pielka, Rajkumar Ramamurthy, Tim Dilmaghani, Bernd Kliem, Rüdiger Loitz, and Rafet Sifa. 2022. Zero-shot text matching for automated auditing using sentence transformers. *Computation and Language*.

Qingqing Cao, Aruna Balasubramanian, and Niranjan Balasubramanian. 2020. Towards accurate and reliable energy measurement of NLP models. *CoRR*, abs/2010.05248.

Qian Chen, Zhen-Hua Ling, and Xiaodan Zhu. 2018. Enhancing sentence embedding with generalized pooling. *Computation and Language*.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.

Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. 2022. On the effectiveness of parameter-efficient fine-tuning.

Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. 2019. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*.

Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. 2020. Towards the systematic reporting of the energy and carbon footprints of machine learning. *CoRR*, abs/2002.05651.

Dan Hendrycks, Kimin Lee, and Mantas Mazeika. 2019. Using pre-training can improve model robustness and uncertainty. *CoRR*, abs/1901.09960.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. *CoRR*, abs/1902.00751.

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

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–133.

Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. 2022. Fine-tuning can distort pretrained features and underperform out-of-distribution. *arXiv:2202.10054*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *CoRR*, abs/2104.08691.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *CoRR*, abs/2101.00190.

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

Jonathan Mamou, Hang Le, Miguel Del Rio, Cory Stephenson, Hanlin Tang, Yoon Kim, and SueYeon Chung. 2020. Emergence of separable manifolds in deep language representations.

John Miller, Rohan Taori, Aditi Raghunathan, Shiori Sagawa, Pang Wei Koh, Vaishaal Shankar, Percy Liang, Yair Carmon, and Ludwig Schmidt. 2021. Accuracy on the line: On the strong correlation between out-of-distribution and in-distribution generalization. *CoRR*, abs/2107.04649.

Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2021. Recent advances in natural language processing via large pre-trained language models: A survey. *CoRR*, abs/2111.01243.

Pramod Kaushik Mudrakarta, Mark Sandler, Andrey Zhmoginov, and Andrew G. Howard. 2018. K for the price of 1: Parameter efficient multi-task and transfer learning. *CoRR*, abs/1810.10703.

Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. *arXiv:1903.05987*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084.

David Rolnick, Priya L. Donti, Lynn H. Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, Alexandra Luccioni, Tegan Maharaj, Evan D. Sherwin, S. Karthik Mukkavilli, Konrad P. Körding, Carla P. Gomes, Andrew Y. Ng, Demis Hassabis, John C. Platt, Felix Creutzig, Jennifer T. Chayes, and Yoshua Bengio. 2019. Tackling climate change with machine learning. *CoRR*, abs/1906.05433.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. 2019. Superglue: Learning feature matching with graph neural networks. *CoRR*, abs/1911.11763.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2019. Green AI. *CoRR*, abs/1907.10597.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. *CoRR*, abs/1906.02243.

Neil C. Thompson, Kristjan H. Greenewald, Keeheon Lee, and Gabriel F. Manso. 2020. The computational limits of deep learning. *CoRR*, abs/2007.05558.

Eduard Tulchinskii, Kristian Kuznetsov, Laida Kushnareva, Daniil Cherniavskii, Serguei Barannikov, Irina Piontkovskaya, Sergey Nikolenko, and Evgeny Burnaev. 2023. Intrinsic dimension estimation for robust detection of ai-generated texts.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018a. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461.

Ran Wang, Haibo Su, Chunye Wang, Kailin Ji, and Jupeng Ding. 2019. To tune or not to tune? how about the best of both worlds? *arXiv:1907.05338*.

Appendix A Hyperparameter Search

We do a hyperparameter grid search on the following parameters. In selecting the number of epochs, we do early stopping based on dev set validation loss for a maximum of 50 epochs. Additionally, we utilize the default train/dev/test splits each task within the GLUE benchmark.

Hyperparameter	Values
Number of epochs	50
Batch size	$32, 512$
Initial learning rate	$1 \times 10^{-2}, 1 \times 10^{-3}$
Hidden layer size	$X/4, X/2, X, 2X, 4X$
Number of layers	$1, 3, 5$
Weight decay	$1 \times 10^{-2}, 1 \times 10^{-4}$
Patience	3

Table 6: Hyperparameter grid search values. X represents the hidden layer size, equal to 768 for all of our methods

Appendix B Description of GLUE tasks

The tasks included are

- CoLA (Corpus of Linguistic Acceptability): Evaluates a model's ability to determine the grammatical acceptability of sentences.
- SST-2 (Stanford Sentiment Treebank - Binary): Involves predicting sentiment labels (positive or negative) for movie reviews.
- MRPC (Microsoft Research Paraphrase Corpus): Requires identifying whether pairs of sentences are semantically equivalent or not.
- QQP (Quora Question Pairs): Focuses on determining duplicate or semantically similar questions.
- STS-B (Semantic Textual Similarity Benchmark): Involves predicting the similarity score between pairs of sentences.
- MNLI (Multi-Genre Natural Language Inference): Requires determining the logical relationship between a premise and a hypothesis in three categories: entailment, contradiction, or neutral.
- QNLI (Question Natural Language Inference): Transforms questions from SQuAD into binary sentence pair classification tasks that require determining whether a certain sentence contains the answer to a given question.
- RTE (Recognizing Textual Entailment): Requires determining if a hypothesis can be inferred from a given premise.
- WNLI (Winograd NLI): Tests a model's ability to resolve pronouns in a sentence.

Appendix C Use of Scientific Artifacts

Our work utilizes the GLUE benchmark dataset, as introduced by Wang et al. (2018a). The dataset is entirely in English and consists of a collection of publicly available text datasets for evaluating natural language understanding systems. It includes tasks such as textual entailment, sentiment analysis, and question answering. The GLUE benchmark ensures that any personally identifiable information is removed from the dataset, making it publicly available for research purposes.

Furthermore, FAST employs Hugging Face implementations for *distilroberta-base*, *all-distilroberta-v1*, *MPNet*, *all-mpnet-base-v2*, all of which are made publicly available for research purposes under the Apache-2.0 license.

Appendix D t-SNE plots

t-SNE plots of raw STSB embeddings:

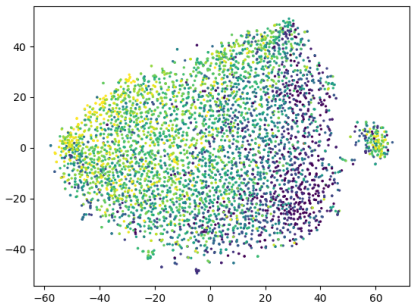

Figure 4: t-SNE plot of STSB embeddings from Mean Pooling

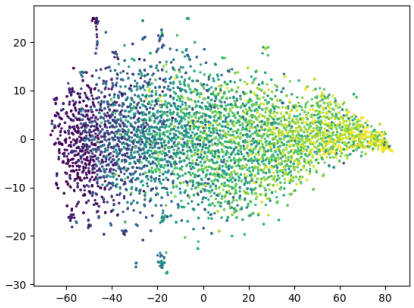

Figure 5: t-SNE plot of STSB embeddings from ST

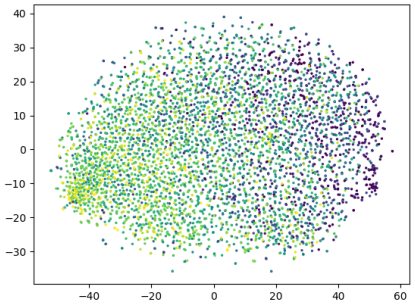

Figure 6: t-SNE plot of STSB embeddings from CBoW

Appendix E GLUE Validation Dev Set

	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	WNLI
CLS + FFN	43.7	87.2	77.7 / 77.7	74.3/74.2	79.7	67.5/67.5	69.9	**57.4**	56.3
Mean Pool + FFN	**52.0**	**89.0**	**79.2/79.2**	**80.2/80.1**	**82.8/82.8**	**68.3/68.3**	**75.3**	57.0	56.3
ST + FFN	21.7	87.0	75.5/75.5	88.6/88.2	80.1/85.2	66.7/66.7	75.2	55.2	56.3
DistilRoBERTa	59.3	92.5	86.6	88.3	89.4	84.0	90.8	67.9	-
Adapters	-	-	-	-	-	-	-	-	-
CBoW	0.0	80.0	73.4/81.5	61.2/58.7	79.1/51.4	56.0/56.4	72.1	54.1	65.1

Table 7: GLUE Benchmark scores (MCC, F1, accuracy) of the CLS, Mean Pool, and Sentence Transformer methods, and comparisons with other methods. CoLA is evaluated using MCC. MRPC and QQP are evaluated using F1 score/accuracy. STS-B uses Spearman's correlation coefficient/accuracy. The other tasks use accuracy.