Enhancing Masked Language Modeling in BERT Models Using Pretrained Static Embeddings

Adam Mištera[1][0009−0000−1019−9218] and Pavel Král[2][0000−0002−3096−675X]

[1] Department of Computer Science and Engineering, Faculty of Applied Sciences, University of West Bohemia, Czech Republic
amistera@kiv.zcu.cz
[2] NTIS – New Technologies for the Information Society, Faculty of Applied Sciences, University of West Bohemia, Czech Republic
pkral@kiv.zcu.cz

Abstract. This paper explores the integration of pretrained static fastText word vectors into a simplified Transformer-based model to improve its efficiency and accuracy. Despite the fact that these embeddings have been outperformed by large models based on the Transformer architecture, they can still contribute useful linguistic information, when combined with contextual models, especially in low resource or computationally constrained environments.
We demonstrate this by incorporating static embeddings directly into our own $\text{BERT}_{\text{TINY}}$-based models prior to pretraining using masked language modeling. In this paper, we train the models on seven different languages covering three distinct language families. The results show that the use of static fastText embeddings in these models not only improves convergence for all tested languages, but also significantly improves their evaluation accuracy.

Keywords: transformers · embeddings · pretraining

1 Introduction

In recent years, models based on the Transformer architecture have become the dominant approach to solving many natural language processing tasks. Although they achieve high accuracy, their practical deployment on devices with limited computational resources is often impossible.

For this reason, more compact variants have emerged that reduce the number of parameters and thus the computational power and memory requirements. However, this reduction in model size usually results in a decrease in model accuracy due to the limited representational capacity of the model.

In this paper, we propose a way to solve this problem by incorporating pretrained embeddings into a small Transformer model. Our goal is to combine the advantages of static embeddings - low computational complexity and language generalization - with the contextual representation offered by the Transformer model. We hypothesize that initializing the model using pretrained embeddings

will allow a small model to better learn the representation using fewer resources without increasing model complexity.

To validate the generality of our approach, we conduct experiments in seven languages from three language families. We show that incorporating static word embeddings not only improves pretraining efficiency, but also consistently improves accuracy during evaluation, especially for morphologically diverse languages.

The main contributions of this paper are:

- a simple architecture that integrates pretrained embeddings into a Transformer model without modifying its training process,
- evaluation across seven different languages from three language families,
- validation of the approach on the masked language modeling objective,
- detailed analysis of model converge, training loss and evaluation accuracy.

The remainder of this paper is structured as follows: Section 2 reviews related work on lightweight Transformers and static embeddings. Section 3 presents our proposed hybrid embedding approach. Section 4 describes the experimental setup, including datasets and training configuration. Section 5 analyzes model convergence and accuracy results trained on multiple languages. Finally, Section 6 concludes the paper and outlines future directions.

2 Related Work

Word embeddings based on the distributional hypothesis [11] represent a way to create a high-quality vector representation of words that can be utilized in many subsequent models to solve downstream tasks such as sentiment analysis [6], named entity recognition [15], or other text classification tasks, e.g., metaphor detection [26]. These embeddings, including Word2vec [18], GloVe [19], and fastText [4], played a key role in many applications of natural language processing (NLP) and still today allow these problems to be solved in low-resource environments due to their speed and memory efficiency.

Compared to static embeddings, attention-based models [24] represent a different approach that uses context-aware embeddings. Models based on attention referred to as *Transformers*, such as BERT [7], RoBERTa [16], ELECTRA [5], and most recently ModernBERT [25] achieved state of the art in many areas of NLP. Parallel to these complex, high-parameter models, smaller versions were also created through knowledge distillation and architecture simplification. These models include TinyBERT [13], DistilBERT [21], $\text{BERT}_{\text{TINY}}$ [23] and MobileBERT [22]. These approaches demonstrate the fact that there is still a need for efficient models with fewer parameters that can solve the same problems as large models.

Current research shows that improvements can be achieved in text classification by using static embeddings and attention-based models [1,9]. These approaches, however, do not include the smaller models mentioned above and do

not consider the impact of using static embeddings directly in the pretraining of Transformer model.

This paper addresses this gap by integrating static pretrained embeddings into $\text{BERT}_{\text{TINY}}$ model and evaluating the impact on both model accuracy and computational efficiency. The mentioned model was chosen because of its rather simple architecture, which was suitable for our experiments. We chose pretrained fastText embeddings, as they yielded better results when used in various deep learning models [8,12]. Specifically, we chose the latest available fastText [10] model pretrained on Common Crawl dataset.

3 Proposed Method

Transformer models, including $\text{BERT}_{\text{TINY}}$ use the *attention* [24] mechanism to learn and better understand the relationships between words or, more precisely, tokens in a given text. It is the attention that makes these models achieve state-of-the-art performance. The calculation of attention can be seen in Equation 1.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}, \tag{1}$$

where $\mathbf{Q} \in \mathbb{R}^{m \times d_k}$, $\mathbf{K} \in \mathbb{R}^{m \times d_k}$, and $\mathbf{V} \in \mathbb{R}^{m \times d_v}$ are the query, key, and value matrices, respectively. Here, d_k is the dimensionality of both queries and keys, d_v is the dimensionality of values, and m is the sequence length.

However, in order for attention to work properly, it is still necessary to create a high-quality representation of the tokens that are its input. BERT-like models learn the input representation during pretraining and the learned embeddings consist of three parts namely token embeddings, position embeddings, and segmentation embeddings as we can see in Equation 2.

$$E = E^{token} + E^{position} + E^{segment} \tag{2}$$

To improve and simplify the training of the model, we propose to replace the randomly initialized E^{token} embeddings with static pretrained embeddings before the start of pretraining as we can see in Equation 3:

$$\vec{e_t} = \begin{cases} \mathcal{E}(t), & \text{t} \in \text{vocabulary} \\ f(t), & \text{otherwise,} \end{cases} \tag{3}$$

where $\vec{e_t}$ denotes the newly obtained vector representation for token t in the tokenizer dictionary of the model. The symbol $\mathcal{E}(t)$ denotes the representation obtained from the pretrained embeddings, and $f(t)$ the fallback strategy if the token is not in the dictionary of the pretrained embeddings. We will discuss the challenges that need to be addressed during token mapping in detail in the following section.

To the best of our knowledge, no previous work has explored the direct integration of static pretrained embeddings into $\text{BERT}_{\text{TINY}}$ prior to pretraining.

We believe that this integration allows $\text{BERT}_{\text{TINY}}$ to benefit from pretrained static representations and thereby improve training efficiency while maintaining the same number of training parameters and model training time.

3.1 Token Alignment

In order to use pretrained word embeddings such as fastText in a model based on the Transformer architecture, several problems need to be solved beforehand. The first, and probably the most important, is the inability to accurately map embedding tokens to the tokens used by the model's tokenizer. This token inconsistency stems from the fact that for training word vectors, the input text is tokenized to whole words. While some may use subword information during the training, they are still stored as whole words. In contrast, BERT models use *WordPiece* tokenization, where words are first split into individual letters and then incrementally merged to form entire words or subwords.

The second issue is the difference in the dimensionality between the pretrained embeddings and the embeddings used by the BERT model. This problem can be solved by adjusting the hidden size of the model to match the dimension of the word vectors. In cases where we do not want to modify the hidden size, it may be better to add a linear mapping to the model, which the model learns during training, but at the cost of slower convergence and an increased number of parameters.

The third one is the difference between the distribution of the word vectors and the initial random initialization in the BERT model used during pretraining. However, this difference can be reduced quite easily by introducing a suitable normalization before using the pretrained word vectors in the model.

4 Experimental Setup

Experiments were conducted on seven languages, namely English (En), German (De), Czech (Cs), Polish (Pl), Italian (It), Spanish (Es) and French (Fr). These languages were selected from a total of three different language groups, namely Germanic, Romance and Slavic, and with varying morphological complexity in order to assess how language type affects the resulting model accuracy.

A custom tokenizer with a vocabulary of 30 000 words was trained for each language. Each tokenizer was trained with 5 million examples and both cased and uncased variants were created for all languages. Initial experiments showed that better results were achieved with the cased variant, so it was chosen for subsequent experiments.

4.1 Model Selection

For our experiments, we chose the $\text{BERT}_{\text{TINY}}$ [3,23] model due to its compact architecture and low number of parameters, which allows significantly faster training than the larger $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$ models even on low-resource

hardware. We then slightly modified the architecture of the selected model, in particular, we increased the number of *attention heads* from 2 to 4, which allows the model to capture a wider range of semantic relationships by focusing on more parts of the input sequence simultaneously, allowing a richer understanding of the input data [14]. This modification of the model increases its representational capacity while keeping it lightweight. At the same time, we also increased the *hidden size* of the model from 128 to 300 to make it easier to integrate static embeddings without changing their dimension, as discussed in detail in Section 3.1. For efficient training on a single GPU, we reduced the *sequence length* from 512 to 128, as suggested in [5] and used to train the ELECTRA-Small model. The complete list of all model parameters is given in the Table 4.

Our main goal was to improve this lightweight architecture without decreasing its efficiency benefits, making it a natural candidate for integrating static embeddings.

4.2 Datasets

The publicly available `allenai/c4` corpus [20] was used to create the training and evaluation datasets. The corpus contains several TB of data, so we chose subsets that were sufficient to train each language for a given number of steps and batch size. For model evaluation, we selected 100 000 examples for each language, with the exception of Czech, whose evaluation dataset contained only 60 462 examples. All created datasets were then preprocessed using the respective tokenizers and truncated to the maximum length of the model context, i.e. 128 tokens.

Table 1: Number of static embeddings vectors found for each language.

EN	DE	CS	PL	ES	IT	FR
21 233	16 595	17 208	16 066	19 147	18 710	19 045

Subsequently, it was necessary to extract static embedding vectors for tokens from the tokenizer vocabulary for each language, as described in Section 3.1. We selected pretrained fastText vectors with the dimension 300, which were first preprocessed, namely normalized and mean-centered, as recommended in [2]. For each language, we searched for word vectors in the top 300 000 most frequent words since we wanted to use only vectors with a high-quality representation. This is because our first experiments showed that the quality of the vectors is a more important factor than the number of vectors. For this reason, it was not possible to find vectors for all the tokens in the dictionary, and therefore the model had to learn the remaining vectors itself. The total number of tokens found for each language is shown in Table 1.

In our experiments, we also analyzed the impact of the different vector initialization options for tokens missing in static embeddings. In particular, we

focused on the initialization using a vector of zeros and random initialization. The results showed that it is better to initialize the vectors of missing tokens using zero vectors. In the case of random initialization, the representations of these tokens were harder for the model to relearn and the convergence of the training loss took noticeably longer.

4.3 Training Configuration

All models were trained for the same number of epochs and used the same optimization strategy to ensure fair comparison. We used the AdamW [17] optimizer, which is most commonly used for training Transformer models [27].

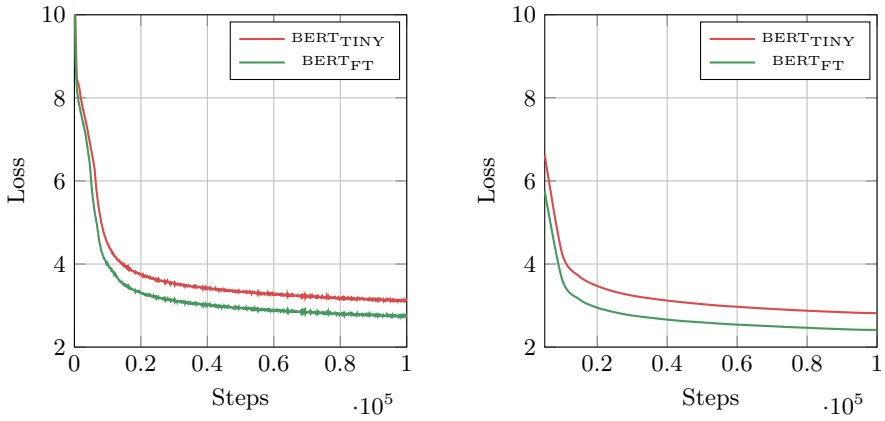
The main hyperparameters were selected based on commonly used configurations for training small Transformer models, with consideration for resource-constrained hardware. We used a learning rate of 5e-4 as used in [5], a batch size of 128, which balances model generalization and GPU memory size requirements, and a maximum sequence length of 128 as mentioned in the model Section 4.1. The total number of training steps was set to 100 000, which provides a reasonable trade-off between model convergence and training time and allows the entire pretraining to be completed in less than 24 hours even on older hardware with a single GPU. In addition, we applied a 10% warmup rate and set the dropout rate to 0.1. The complete list of hyperparameters is provided in table 4.

Pretraining of the model was conducted using the well-known *masked language modeling* (MLM) objective following the procedure described in the original BERT article [7]. Specifically, 15% of the input tokens were randomly masked before training and the model was optimized to predict their original identity. This self-supervised approach allows the model to learn contextual representations across languages without requiring labeled data. We decided not to use the *next sentence prediction* (NSP) objective given the findings in [16], which showed that removing the NSP in the RoBERTa model does not harm performance and may even lead to improvements. Consequently, segmentation embeddings, which are mainly used in the NSP task to distinguish between pairs of sentences, were also omitted. For the implementation we used the Python programming language and the *transformers* library with PyTorch. Furthermore, we fixed the random seed with the same value to make the individual results more reproducible.

5 Results

In this section we focus on the evaluation of the experimental results, in particular the progression of training and evaluation loss during training and the resulting accuracy of the models measured on the evaluation dataset. For training and evaluation loss, we report cross-entropy loss values, and accuracy represents the number of correctly identified masked tokens.

The Figure 1a shows the training loss during training of the Czech model. Czech language was chosen for the exemplar plot because it is a morphologically very rich language and, as might be expected, the use of embeddings is more



(a) Train loss (b) Evaluation loss

Fig. 1: Training and evaluation loss measured during pretraining of the Czech language model. The value of evaluation loss was computed every 5 000 steps.

apparent in its case. In the plot, $\text{BERT}_{\text{TINY}}$ refers to the model described in Section 4.1 and BERT_{FT} denotes our model using pretrained fastText embeddings. The plot shows that for the model using pretrained embeddings, its loss function decreases faster, i.e. the model learns more easily. At the same time, training of our model is very efficient, as after less than a third of the total training time, BERT_{FT} reaches a lower value of the loss function than $\text{BERT}_{\text{TINY}}$ after the whole training.

Table 2: Train and evaluation loss across all languages.

Model	**En**	**De**	**Cs**	**Pl**	**Es**	**It**	**Fr**
Training dataset loss values							
$\text{BERT}_{\text{TINY}}$	3.459	3.467	3.125	3.266	3.174	3.317	3.162
BERT_{FT}	**3.272**	**3.098**	**2.750**	**3.115**	**2.867**	**2.917**	**2.881**
Evaluation dataset loss values							
$\text{BERT}_{\text{TINY}}$	3.213	3.160	2.818	2.819	2.871	3.042	2.897
BERT_{FT}	**2.992**	**2.729**	**2.414**	**2.427**	**2.550**	**2.637**	**2.606**

The Figure 1b shows the progression of the loss function measured on the evaluation dataset. As in the previous case, the increased training efficiency and the overall lower value of the loss function of our model compared to the baseline $\text{BERT}_{\text{TINY}}$ is evident.

Table 2 shows the training and evaluation loss across seven languages. The proposed method consistently outperforms the standard $\text{BERT}_{\text{TINY}}$ model, with the largest gains observed in German, Italian, Czech and Polish.

5.1 Accuracy

In addition to the training and evaluation losses of the model, the resulting accuracy is also important, so we focus on its assessment in detail in this section. As in the previous case, we again focus on the detailed evaluation of the Czech model. The detailed progress of the accuracy during training can be seen in Figure 2. The accuracy of the model was evaluated on an evaluation dataset every 5000 steps. The graph shows that after only 20 000 steps, our proposed model already achieves higher accuracy than the classical model after training. Overall, after training, BERT_{FT} achieves more than seven percent higher accuracy than $\text{BERT}_{\text{TINY}}$. Therefore, the use of static embeddings also significantly improves the accuracy of the resulting model. The progress of accuracy during training for the remaining tested languages can be seen in Figure 3.

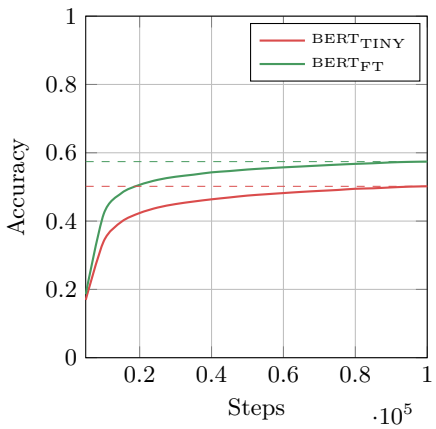


Fig. 2: Accuracy of the Czech model during training assessed on the evaluation dataset. The accuracy of the model was computed every 5 000 steps.

In Table 3, we can see the accuracy measured after training for all included languages. On average, the increase in accuracy of our proposed model over all languages is around six percent. In particular, the involvement of static embeddings helped German, Italian, Czech and Polish the most, with an improvement of more than seven percent. As we anticipated, the highest increases occurred especially in the morphologically richer languages. In comparison, the smallest increase was observed in English, more specifically three percent.

Table 3: Evaluation accuracy across all tested languages.

Model	EN	DE	CS	PL	ES	IT	FR
$\text{BERT}_{\text{TINY}}$	0.436	0.452	0.502	0.499	0.487	0.461	0.488
BERT_{FT}	**0.468**	**0.526**	**0.574**	**0.569**	**0.544**	**0.534**	**0.540**

6 Conclusion

This paper presents a simple yet effective approach to improve the accuracy and efficiency of $\text{BERT}_{\text{TINY}}$ by initializing its token embeddings with pretrained fastText word vectors prior to pretraining. Our experiments show that the model converges faster during pretraining while achieving consistent accuracy gains across all seven tested languages, and maintaining nearly the same model size. The average improvement of six percent suggests that the proposed approach is suitable for different languages.

We believe that combining static and contextual embeddings is a promising direction for future low-resource NLP applications. In future work, we would like to focus also on multilingual models. For these models, we would also apply linear transformations, which would map the word embeddings used to the same space so that they could be used in the model.

Acknowledgement

This work has been supported by the Grant No. SGS-2025-022 – New Data Processing Methods in Current Areas of Computer Science. Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

1. Alghanmi, I., Espinosa Anke, L., Schockaert, S.: Combining BERT with static word embeddings for categorizing social media. In: Xu, W., Ritter, A., Baldwin, T., Rahimi, A. (eds.) Proceedings of the Sixth Workshop on Noisy User-generated Text (W-NUT 2020). pp. 28–33. Association for Computational Linguistics, Online (Nov 2020)
2. Artetxe, M., Labaka, G., Agirre, E.: Learning principled bilingual mappings of word embeddings while preserving monolingual invariance. In: Su, J., Duh, K., Carreras, X. (eds.) Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. pp. 2289–2294. Association for Computational Linguistics, Austin, Texas (Nov 2016)
3. Bhargava, P., Drozd, A., Rogers, A.: Generalization in nli: Ways (not) to go beyond simple heuristics (2021)
4. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics **5**, 135–146 (2017)

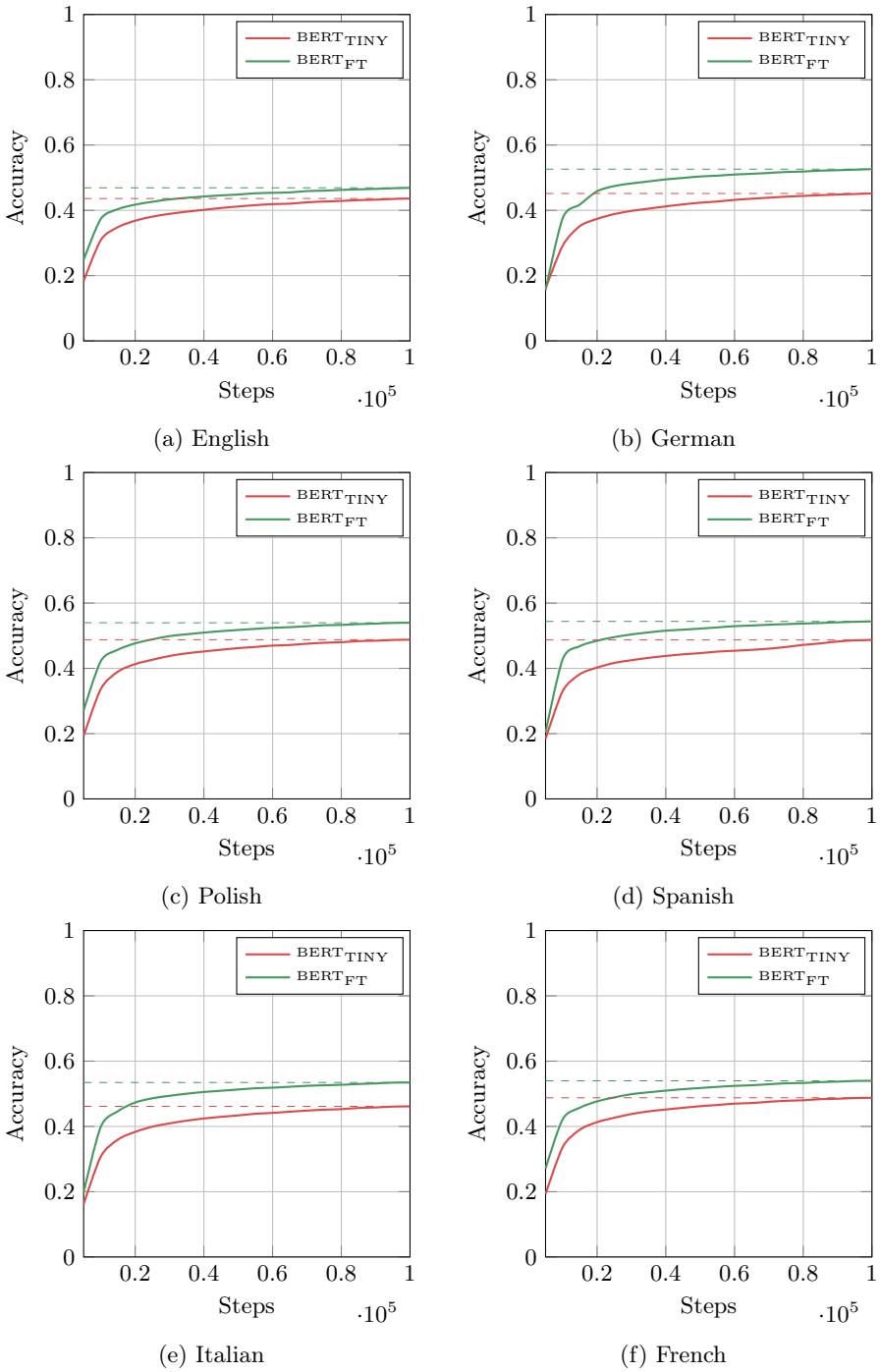
5. Clark, K., Luong, M., Le, Q.V., Manning, C.D.: ELECTRA: pre-training text encoders as discriminators rather than generators. CoRR **abs/2003.10555** (2020)
6. Cliche, M.: BB_twtr at SemEval-2017 task 4: Twitter sentiment analysis with CNNs and LSTMs. In: Bethard, S., Carpuat, M., Apidianaki, M., Mohammad, S.M., Cer, D., Jurgens, D. (eds.) Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017). pp. 573–580. Association for Computational Linguistics, Vancouver, Canada (Aug 2017)
7. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Burstein, J., Doran, C., Solorio, T. (eds.) Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (Jun 2019)
8. Dharma, E.M., Gaol, F.L., Warnars, H., Soewito, B.: The accuracy comparison among word2vec, glove, and fasttext towards convolution neural network (cnn) text classification. J Theor Appl Inf Technol **100**(2), 31 (2022)
9. D'Sa, A.G., Illina, I., Fohr, D.: Bert and fasttext embeddings for automatic detection of toxic speech. In: 2020 International Multi-Conference on: "Organization of Knowledge and Advanced Technologies" (OCTA). pp. 1–5 (2020)
10. Grave, E., Bojanowski, P., Gupta, P., Joulin, A., Mikolov, T.: Learning word vectors for 157 languages. In: Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018) (2018)
11. Harris, Z.: Distributional structure. Word **10**(23), 146–162 (1954)
12. Ilham, M.R., Laksito, A.D.: Comparative analysis of using word embedding in deep learning for text classification. Jurnal Riset Informatika **5**(2), 195–202 (2023)
13. Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., Liu, Q.: Tinybert: Distilling bert for natural language understanding. arXiv preprint arXiv:1909.10351 (2019)
14. Jo, J.y., Myaeng, S.H.: Roles and utilization of attention heads in transformer-based neural language models. In: Proceedings of the 58th annual meeting of the association for computational linguistics. pp. 3404–3417 (2020)
15. Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., Dyer, C.: Neural architectures for named entity recognition. In: Knight, K., Nenkova, A., Rambow, O. (eds.) Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 260–270. Association for Computational Linguistics, San Diego, California (Jun 2016)
16. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019)
17. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017)
18. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013)
19. Pennington, J., Socher, R., Manning, C.: GloVe: Global vectors for word representation. In: Moschitti, A., Pang, B., Daelemans, W. (eds.) Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1532–1543. Association for Computational Linguistics, Doha, Qatar (Oct 2014)
20. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. CoRR **abs/1910.10683** (2019)

21. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108 (2019)
22. Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., Zhou, D.: Mobilebert: a compact task-agnostic bert for resource-limited devices. arXiv preprint arXiv:2004.02984 (2020)
23. Turc, I., Chang, M., Lee, K., Toutanova, K.: Well-read students learn better: The impact of student initialization on knowledge distillation. CoRR **abs/1908.08962** (2019)
24. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)
25. Warner, B., Chaffin, A., Clavié, B., Weller, O., Hallström, O., Taghadouini, S., Gallagher, A., Biswas, R., Ladhak, F., Aarsen, T., Cooper, N., Adams, G., Howard, J., Poli, I.: Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference (2024)
26. Wu, C., Wu, F., Chen, Y., Wu, S., Yuan, Z., Huang, Y.: Neural metaphor detecting with CNN-LSTM model. In: Beigman Klebanov, B., Shutova, E., Lichtenstein, P., Muresan, S., Wee, C. (eds.) Proceedings of the Workshop on Figurative Language Processing. pp. 110–114. Association for Computational Linguistics, New Orleans, Louisiana (Jun 2018)
27. Zhuang, B., Liu, J., Pan, Z., He, H., Weng, Y., Shen, C.: A survey on efficient training of transformers. arXiv preprint arXiv:2302.01107 (2023)

A Appendix

Table 4: Pretraining hyperparameters of the $\text{BERT}_{\text{TINY}}$ model.

Hyperparameter	**Value**
Number of Layers	2
Number of Attention Heads	4
Hidden Size	300
Intermediate Size	512
Maximum Sequence Length	128
Vocabulary Size	30000
Number of Parameters	10501324
Dropout Rate	0.1
Learning Rate	5e-4
Batch Size	128
Number of Steps	100000
Warmup steps	10000
Optimizer	AdamW



(a) English (b) German

(c) Polish (d) Spanish

(e) Italian (f) French

Fig. 3: Evaluation accuracy during pretraining for the tested language.