pNLP-Mixer: an Efficient all-MLP Architecture for Language

Francesco Fusco
IBM Research
ffu@zurich.ibm.com

Damian Pascual[*]
Telepathy Labs
damian.pascual@telepathy.ai

Peter Staar
IBM Research
taa@zurich.ibm.com

Diego Antognini
IBM Research
Diego.Antognini@ibm.com

Abstract

Large pre-trained language models based on transformer architecture have drastically changed the natural language processing (NLP) landscape. However, deploying those models for on-device applications in constrained devices such as smart watches is completely impractical due to their size and inference cost. As an alternative to transformer-based architectures, recent work on efficient NLP has shown that weight-efficient models can attain competitive performance for simple tasks, such as slot filling and intent classification, with model sizes in the order of the *megabyte*. This work introduces the pNLP-Mixer architecture, an embedding-free MLP-Mixer model for on-device NLP that achieves high weight-efficiency thanks to a *novel projection layer*. We evaluate a pNLP-Mixer model of only *one megabyte* in size on two multi-lingual semantic parsing datasets, MTOP and multiATIS. Our quantized model achieves 99.4% and 97.8% the performance of mBERT on MTOP and multi-ATIS, while using *170x fewer parameters*. Our model consistently beats the state-of-the-art of tiny models (pQRNN), which is twice as large, by a margin up to 7.8% on MTOP.

1 Introduction

Large language models based on transformer architecture have been fueling the latest successes in natural language processing (NLP). Nowadays, fine-tuning pre-trained models represents the defacto framework to tackle diverse NLP tasks, even those with limited amounts of annotations.

While the merit of large pre-trained language models is undeniable, using models of several gigabytes and billions of parameters is not always practical or even possible due to computational and memory requirements. In addition, there are many simple and yet important tasks, such as slot filling in home assistants, which do not require the complex linguistic knowledge encoded by large pre-trained models and for which smaller models may reach competitive performance at a much lower cost. Reducing model sizes to the order of the *megabyte* is a necessity for resource constrained devices, such as smart watches, and in general it is attractive for edge use cases as i) updating models at the edge requires pushing updates to potentially millions of devices, and ii) multiple models solving different tasks can be deployed even on embedded devices with limited memory capacity.

Transformer-based architectures are not suitable to downscale to such *ultra small* model sizes, mostly due to the space required to store embedding tables (Zhao et al., 2021). Projection-based models (Ravi, 2017) have shown that the dense representations learned as part of the training process and stored in the embedding tables can be replaced by non-trainable representations computed on-the-fly over the text, hence the name embedding-free.

In this work, we introduce the *pNLP-Mixer*, a novel embedding-free architecture for ultra-small NLP models targeting on-device applications. Our architecture relies on a novel *projection layer* which creates text representations for individual tokens by combining the MinHash fingerprints (Broder, 2000) corresponding to each subword unit. The projected features are given as input to a MLP-Mixer (Tolstikhin et al., 2021), which grants our model architecture linear scalability in the sequence length and seamless hardware acceleration. To the best of our knowledge, this is the first work combining subword-unit tokenization and MinHash fingerprints in projection networks.

Our evaluation on two semantic parsing datasets representative of on-device applications, MTOP and multiATIS, showcases that the pNLP-Mixer beats the current state-of-the-art for ultra-small models, pQRNN (Kaliamoorthi et al., 2021), by up to 7.8% on sequence tagging tasks. On MTOP,

[*]Work done during a research stay at IBM Research.

a pNLP-Mixer model with only one million parameters achieves 99.4% of the performance of mBERT, which has *170x* more parameters.

2 Related Work

Since the introduction of transformer-based language models such as BERT (Devlin et al., 2019), model sizes have been increasing at unprecedented pace (Brown, 2020; Goyal et al., 2021; Lample and Conneau, 2019). Using current large language models for on-device applications is simply not feasible due to the size and computational requirements, especially in resource constrained devices such as smart watches. Transformer-based models optimized for smartphone use cases, such as DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2020), and MobileBERT (Sun et al., 2020) have shown that by combining knowledge distillation (Hinton et al., 2015) and quantization (Jacob et al., 2018), one can achieve model sizes in the order of tens to hundreds of megabytes in size. Embedded devices, such as wearables, require instead model sizes in the order of the *megabyte*, a target that is very challenging to achieve with transformer-based architecture, mostly because of the size of the embedding tables (Zhao et al., 2021).

Embedding-free model architectures have been introduced to completely eliminate the dependency on large embedding tables from models. Instead of learning embeddings at training time, text representation are computed on-the-fly using solely the surface forms of the tokens by means of locality-sensitive hashing (LSH) (Charikar, 2002) techniques. This way tokens that are similar at the surface level have similar representations. The idea of replacing trainable parameters stored in embedding tables with LSH-based projections has been introduced in Ravi (2017) and Ravi (2019). Follow up research work on model architectures targeting ultra-small model sizes has resulted in several model architectures including SGNN (Ravi and Kozareva, 2018), SGNN++ (Ravi, 2019), Prado (Kaliamoorthi et al., 2019), and pQRNN (Kaliamoorthi et al., 2021). Our model architecture belongs to the same line of research, but introduces a linguistically informed projection layer which combines subword-unit tokenization (Sennrich et al., 2016) with LSH principles. In our work, we evaluate and compare multiple LSH techniques, including SimHash (Manku et al., 2007) and MinHash (Broder, 2000). In our projection layer, by exploiting the associativity property of MinHash, fingerprints of individual tokens can be efficiently computed from the fingerprints of their subword units.

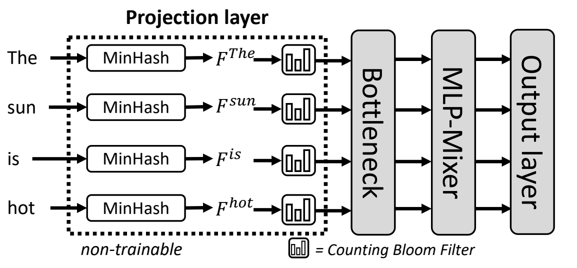

Figure 1: Our pNLP-Mixer model has a *non-trainable* projection layer which feeds a MLP-Mixer architecture with rich text features representing MinHash *fingerprints* in a Counting Bloom Filter.

Our model does not use attention mechanisms, as it feeds the representations to a MLP-Mixer (Tolstikhin et al., 2021) model. While using MLP only architectures is not new in the NLP landscape (Liu et al., 2021; Yu et al., 2022), this work is the first proposing an all-MLP architecture for ultra-small models. There are numerous studies around efficient transformer-based models (Tay et al., 2022) and solutions to make them scale linearly with the sequence length. However, none of those work targets models of the size of the single megabyte.

3 pNLP-Mixer: a Projection MLP-Mixer

The pNLP-Mixer has been designed from the ground up as an efficient architecture suitable for both edge cases, memory and latency constrained, and as a backbone for complex NLP pipelines.

Figure 1 depicts the model architecture at high level. The pNLP-Mixer falls into the category of projection-based models: instead of storing large embedding tables, like transformer-based models do, our model uses a projection layer which captures morphological knowledge from individual tokens using *non-trainable* hash functions. This projection layer can be seen as a feature extractor that produces a representation from input text. Once the input features are computed, they are passed through a trainable linear layer called bottleneck layer. The output of the bottleneck layer is the input of a series of MLP blocks of a standard MLP-Mixer architecture (Tolstikhin et al., 2021).

There are several advantages of using an all-MLP architecture for language processing. In contrast to attention-based models, the MLP-Mixer captures long-range dependencies without introducing a quadratic cost on the sequence length. Further, by using only MLPs, the model becomes simple to implement and has out-of-the-box hardware acceleration in devices ranging from mobile phones to server-grade inferencing accelerators.

The main contribution of our work is to show that a simple model like the MLP-Mixer represents a valid alternative to transformer-based models in NLP, even in setups where large embedding tables are replaced with projections computed on the fly. The key to achieve competitive performance with such small and computationally efficient models is to feed them with high-quality input features.

3.1 Projection Layer

Our projection layer builds upon the notion of locality sensitive hashing (LSH) (Indyk and Motwani, 1998) to create representations from text. While LSH has been introduced in previous works, e.g., in pQRNN (Kaliamoorthi et al., 2021), our approach is completely novel. In particular, we combine subword-unit tokenization (Schuster and Nakajima, 2012; Sennrich et al., 2016) and the *associativity* of MinHash (Broder, 2000) to efficiently compute features of any token as a combination of the features corresponding to its subword unit. Subword tokenization, which is commonly used in transformers, ensures that any text can be represented as a sequence of subwords units, i.e., there are no out-of-vocabulary words. In our context, using subword tokenization provides two main advantages: i) linguistic knowledge can be injected by training domain-specific subword-unit tokenizers, and ii) the representation of each subword unit can be precomputed and cached to reduce inference costs.

Our projection layer calculates the MinHash fingerprint F^t of each input token t by reusing the fingerprint of individual subword units belonging to the vocabulary V (see Figure 2). A fingerprint $F \in \mathbb{N}^n$ is an array of n positive integers F_0 to F_{n-1}, computed with n distinct hash functions $h_0(x)$ to $h_{n-1}(x)$ mapping strings to positive integers. This way, the first step of our projection is tokenization, which transforms each input token into a list of subword units. Then, for each subword unit u, we calculate its fingerprint F^u. Each element F_i^u, with $0 \leq i < n$, is obtained by first applying a hash function $h_i(x)$ to each of the trigrams v_0 to v_{k-1} extracted from the subword u, with $k \geq 1$. Then, F_i^u is obtained as the minimum hash value across trigrams: $F_i^u = min(h_i(v_0), ..., h_i(v_{k-1}))$. For example, for the subword unit "*Bring*", F_i^{Bring} is computed as F_i^{Bring}=$min(h_i(\text{"}Bri\text{"}), h_i(\text{"}rin\text{"}), h_i(\text{"}ing\text{"}))$.

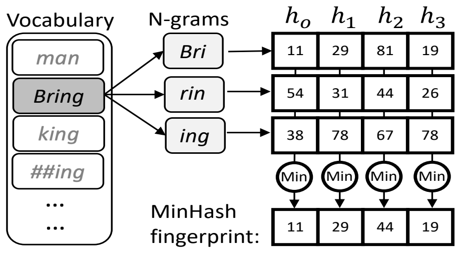

Figure 2: The MinHash fingerprint of a subword unit contains the minimum hash values computed over the trigrams, for each hash function H_{0-3}. Fingerprints for a given token are computed by aggregating the fingerprints of its subword units in a similar way.

When a subword is a continuation, e.g., "*##ing*", we skip the trigram extraction and calculate the hash $h_i(x)$ directly on the full subword unit u. The fingerprint F^u is built by calculating F_i^u for each of the n hash functions $h_0(x)$ to $h_{n-1}(x)$.

Finally, the fingerprint F^t of a token t made of several subword units u_0 to u_{j-1}, e.g., "*Bringing*" $\rightarrow$ ["*Bring*", "*##ing*"], is simply obtained by setting each element F_i^t to the minimum across the j subword fingerprints $F_i^{u_0}$ to $F_i^{u_{j-1}}$. In our example, $F_i^{Bringing} = min(F_i^{Bring}, F_i^{\#\#ing})$.

In practice, if the fingerprint of each subword unit u in the vocabulary V is precomputed and cached, inference does not require *any hashing on strings* but only computing the *minimum* between integer sets. In our setup, we use the minimum operator as described in the original MinHash paper (Broder, 2000), which also contains the required theoretical foundations. What we introduce in our work is a method that elegantly exploits the associativity property of MinHash to avoid computing hash functions over strings at runtime.

For each input token t, we do not use the fingerprints directly as input to the bottleneck layer but, instead, we use them to populate an array $C^t \in \mathbb{R}^m$ of m float counters initially set to zero. In detail, given the fingerprint F^t corresponding to the token t, for each of the n MinHash values F_i^t, we increase by one the value in position p of the array of counters C^t, where $p = F_i^t \ mod \ m$. Therefore, the extracted feature from each token is essentially a Counting Bloom Filter (Fan et al., 2000) storing the set of integers part of its MinHash fingerprint.

Caching fingerprints for subword-units is entirely *optional*. The memory required to enable caching is given by an integer matrix storing $|V|$ fingerprints of size n, where $|V|$ is the vocabulary size and n the number of hash functions. In practice, caching costs just a few megabytes of memory with vocabularies from pre-trained models and fingerprints with, e.g., 64 hashes. Note, that there is a fundamental difference between the embedding matrices stored in transformers and our cached fingerprints. In contrast to embedding tables, which in transformer-based models are task specific as a result of the fine-tuning process, the fingerprints are *not trainable* and they are not directly involved in any matrix multiplication. Since fingerprints are not trainable, they can be reused across different models, i.e., a *single* cache can serve n models. In complex NLP pipelines to be executed on embedded devices at the edge, this architecture provides substantial opportunities for optimizations. First, the same token fingerprint can be reused to perform the inference with distinct models, which means that the cost of computing the projection can be easily amortized. Second, as long as tokenizer and hashing scheme do not change, distinct models can be independently updated, while keeping the cache of subword-unit fingerprints unmodified on the device. Third, having a cache that is shared among models means that the memory costs required to enable caching are also amortized. It is worth to remark that the advantages offered by our architecture are not limited to edge use-cases. Large-scale natural language processing platforms running in data-centers can equally benefit from the resource optimization and granular deployment opportunities offered by our architecture.

3.2 MLP-Mixer

The MLP-Mixer (Tolstikhin et al., 2021) is a simple architecture that consists exclusively of mixer blocks. Each block has two multi-layer perceptrons (MLPs) interleaved by a transposition operation. The transposition of the output of the first MLP lets the second operate on the sequence dimension, effectively mixing information across tokens. Our model follows the original work.

In our case, the matrix $C \in \mathbb{R}^{s \times m}$ produced by the projection layer, where s the sequence length and m the size of the counting bloom filter, is passed through a bottleneck layer: a dense layer followed by an activation function and a normalization layer, that outputs a matrix $B \in \mathbb{R}^{s \times h}$, where h is the hidden size. B is fed to the MLP-Mixer, which in turn produces an output $O \in \mathbb{R}^{s \times h}$. We apply a classification head on top of O to generate the predictions. In the case of semantic parsing this head is a linear layer applied on each token, while for classification tasks, we use a max pooling instead.

4 Experimental Setup

Our architecture is designed as an alternative to transformer-based models for *ultra-small* models (i.e., one megabyte) targeting on-device applications. In the league of extremely small models, common evaluation datasets used by research and industry are not the same as the ones used for evaluating the generalizability of large pre-trained language models (e.g., GLUE (Wang et al., 2018)), but datasets for simpler tasks, that are more realistic applications for tiny models. Thus, we align to prior works on tiny models for on-device applications (Kaliamoorthi et al., 2019, 2021) that assess models on two multilingual semantic parsing datasets.

MTOP (Li et al., 2021). It covers six languages, English, Spanish, French, German, Hindi, and Thai. It was created by translating from English to the other languages. The train, dev, and test set for each language contain 10k, 1.5k, and 3k samples. We assess the models on slot parsing (*named entity recognition*) on 78 different slot labels. We report the exact match accuracy score, computed as the number of instances whose *all* tokens have been correctly labeled over the number of instances.

multiATIS (Xu et al., 2020). It is a multilingual version of the ATIS dataset (Price, 1990), that contains queries related to air travel in nine languages: English, Spanish, French, German, Hindi, Japanese, Portuguese, Turkish, and Chinese. Each language except Hindi and Turkish consists of $4,488/490/893$ samples for train, dev, and test sets; for Hindi and Turkish the splits are $1,440/160/893$ and $578/60/715$. We evaluate on *intent classification*: determining the intent of a query from 18 labels, and we report the accuracy.

Training Details. In our experiments, we aim for model sizes in the order of *one million parameters* (one megabyte with 8-bit quantization). All trained models are approximately of this size. For the pNLP-Mixer, the projection of each token is a feature vector of dimension 512 filled with 256 hashes. The bottleneck consists of one MLP

Projection	MTOP EN Exact Match Acc.	multiATIS EN Intent Acc.
Binary	80.58	97.97
TSP	80.33	98.17
SimHash	80.99	98.38
MinHash (Ours)	**82.51**	**98.57**

Table 1: Comparison of different projection layers followed by the bottleneck and MLP-Mixer on the validation set of English MTOP and multiAtis.

Model	# Param.	MTOP EN Exact Match Acc.	multiATIS EN Intent Acc.
Projection-only	0.2M	49.15	81.54
CNN	1.0M	73.74	97.77
LSTM	1.2M	76.92	97.77
Transformer	1.0M	74.05	97.97
MLP-Mixer	1.0M	**82.51**	**98.57**

Table 2: Comparison of different architectures using the MinHash projection layer on the validation set of English MTOP and multiATIS.

with a Leaky ReLU as the activation function (Xu et al., 2015) followed by a normalization layer (Ba et al., 2016). Finally, we use 5 Mixer layers where each block contains 256 hidden dimensions for the token-mixing, channel-mixing, and classification head. We use the tokenizer of BERT-base multilingual cased. We tune the learning rate, weight decay, and dropout with a batch size of 128 and using early-stopping with a patience of 5 epochs. We select the models reaching the best exact match and intent accuracy on the validation set. We report their performance on the test set.

5 Model Investigation

We provide detailed insights on the impact of different projection layers and other architectural components as well as a comparison to alternative architectures. We perform the experiments on the English variant of the MTOP and multiATIS datasets.

5.1 Projection Comparison

First, given the pNLP-Mixer model of Section 4 with input features fixed to 512, we compare different feature extraction strategies. Specifically:

• **Binary.** We compute 256 hash values for each token. Given a token and a bitmap of size $m = 512$ set to zero, for each hash value h_v, we set to 1 the bit in position $p = h_v \ mod \ m$ of the bitmap. The token feature is a float tensor storing the bitmap.

• **TSP.** For each token a 1024-bits hash is computed and then represented as ternary feature of size 512 as described in Kaliamoorthi et al. (2019).

• **MinHash.** Our projection layer (Section 3.1).

• **SimHash.** We compute the hashes of subword units as in *MinHash*, but we combine them using SimHash (Manku et al., 2007; Shrivastava and Li, 2014). The extracted feature is a binary feature of size l, where l is the size (in bits) of the hashes applied to n-grams or entire subword units. The value at index $0 \leq p < l$ of the feature is the sign of the value ϕ_p stored at index p of a histogram ϕ of length l. The histogram, initialized to 0, is populated by summing or subtracting 1 to ϕ_p whenever a hash value has a 1 or 0 in position p.

In Table 1, we report the best scores obtained after tuning each configuration. Overall, our projection layer MinHash obtains the best exact match accuracy and intent accuracy, with an absolute improvement over SimHash of $+1.52$ and $+0.19$. Binary and TSP obtain the worst performance: -1.93 and -2.18 on the MTOP compared to MinHash, and -0.60 and -0.4 on multiATIS. Those differences confirm the limitation of binary and ternary features and highlight the importance of carefully designing the projection layer and justifies an effort for further research on projection algorithms. Given these results, we only consider our MinHash-based projection for the rest of the experiments.

5.2 Model Comparison

Now, we investigate whether the MLP-Mixer is the optimal architecture to process this representation. First, we remove the MLP-Mixer and connect the output of the bottleneck layer to the classification heads (Projection-Only). Then, we replace the MLP-Mixer with three alternative architectures: a convolutional neural network (CNN) (LeCun et al., 2015), a long short-term memory recurrent neural network (LSTM) (Hochreiter and Schmidhuber, 1997), and a transformer (Vaswani et al., 2017).

Table 2 shows that using the projection-layer directly as input to the classification heads without a model in between, results in very poor performance. From the alternative models, all perform significantly worse than the MLP-Mixer: -8.77, -5.59, and -8.46 in terms of exact match accuracy for the CNN, LSTM, and transformer models, respectively. This last result is remarkable: for the same number of parameters, the MLP-Mixer outperforms the transformer while having a linear complexity on the input length instead of a quadratic one. Overall,

Model	# Param.	Intent Accuracy EN	ES	FR	DE	HI	JA	PT	TR	ZH	Avg
LSTM	28M	96.1	93.0	94.7	94.0	84.5	91.2	92.7	81.1	92.5	91.1
mBERT	170M	<u>98.3</u>	<u>97.4</u>	<u>98.6</u>	<u>98.5</u>	<u>94.5</u>	<u>98.6</u>	<u>97.4</u>	<u>91.2</u>	<u>97.5</u>	<u>96.9</u>
Transformer	2M	96.8	92.1	93.1	93.2	79.6	90.7	92.1	78.3	88.1	89.3
pQRNN	2M$_{(8bit)}$	98.0	97.0	97.9	96.6	**90.7**	88.7	**97.2**	86.2	93.5	94.0
pNLP-Mixer	1M$_{(8bit)}$	**98.1**	**97.1**	**98.1**	**97.3**	**90.7**	**92.3**	**97.2**	**87.3**	**95.1**	**94.8**

Table 3: Intent accuracy across languages on the test sets of multiATIS. For each language we <u>underline</u> the best overall result and we mark in **bold** the best performance among the tiny models.

Model	#Param.	Exact Match Accuracy EN	ES	FR	DE	HI	TH	Avg
XLU	70M	78.2	70.8	68.9	65.1	62.6	68.0	68.9
XLM-R	550M	<u>85.3</u>	81.6	79.4	<u>76.9</u>	<u>76.8</u>	73.8	<u>79.0</u>
mBERT	170M	84.4	<u>81.8</u>	<u>79.7</u>	76.5	73.8	72.0	78.0
Transformer	2M	71.7	68.2	65.1	64.1	59.1	48.4	62.8
pQRNN	2M$_{(8bit)}$	78.8	75.1	71.9	68.2	69.3	68.4	71.9
- distilled		79.4	75.4	73.0	68.6	70.2	69.5	72.7
pNLP-Mixer	1M$_{(8bit)}$	**84.0**	**78.3**	**75.2**	**76.9**	**76.5**	**<u>74.1</u>**	**77.5**

Table 4: Exact match accuracy across languages on the test sets of MTOP. We <u>underline</u> the best overall result for each language and mark in **bold** the best performance among the tiny models.

the evaluation shows that the MLP-Mixer is weight-efficient for processing the projection output and reaching high performance.

6 Evaluation

Finally, we run a complete evaluation on the test sets of MTOP and multiATIS. We compare our pNLP-Mixer with three very large models: XLU (Lai et al., 2019), which is a bi-LSTM model with pretrained XLU embeddings, and two pre-trained multilingual models: XLM-R (Conneau et al., 2020) and multilingual BERT (mBERT) (Devlin et al., 2019). We also include two small models: pQRNN (Kaliamoorthi et al., 2021) and a simple transformer using the same projection as pQRNN. pQRNN is an embedding-free Quasi-RNN (Bradbury et al., 2017) model that shares the same philosophy of our proposed pNLP-Mixer: a small and task-specific model that learns directly from the text. For a fair comparison against pQRNN, we quantize our pNLP-Mixer models and report the performance on the 8-bit version. Finally, we include pQRNN distilled with mBERT on MTOP (the original study did not distill pQRNN on multiATIS). The performance values of all the baselines are taken from Kaliamoorthi et al. (2021).

MTOP. Table 4 shows that the large pre-trained models, XLM-R and mBERT, obtain the highest scores. Notably, from the smaller alternatives, our pNLP-Mixer with only 1M parameters, 8-bit quantization and no pretraining, i.e., *680x smaller than mBERT*, reaches an average exact match accuracy only 0.5 and 1.5 points lower than mBERT and XLM-R. It even beats mBERT in the non-European languages. With those results, the pNLP-Mixer beats a twice larger pQRNN model across all languages by 7.8% in average. It even beats a pQRNN model distilled from mBERT by 6.6% in average.

multiATIS. Table 3 shows a similar trend compared to the MTOP dataset. On average, the pNLP-Mixer performs better than pQRNN while being twice as small. Remarkably, the pNLP-Mixer significantly outperforms the transformer model and the larger LSTM. Moreover, it reaches 97.8% of the performance of mBERT while being 680x smaller.

Discussion. The results show that the pNLP-Mixer represents a very competitive model for the settings where the maximum model size is limited due to either memory or latency requirements. Our pNLP-Mixer models, with only 1M parameters and a size of one megabyte when quantized, reaches competitive scores in both datasets compared to mBERT, which is a 680x larger model. This represents an important step towards ultra-small models for NLP. To put numbers in perspective, for the non-quantized pNLPN-Mixer model, the inference latency with batch size 1 on a *single CPU core* is as little as 2.4ms,[1] with the projection layer taking 0.4ms. Finally, we could not compare pNLP-Mixer with pQRNN in terms of FLOPS or latency because the authors did not make the code available; we are unable to produce comparable predictive performance with our implementation of pQRNN.

[1]We report the average latency across 100 samples on a Xeon E5-2690v4 processor and a PyTorch runtime.

7 Conclusion

We introduce pNLP-Mixer, the first embedding-free model based on the MLP-Mixer architecture. Our main contribution is an efficient and yet effective projection layer which combines MinHash fingerprints and subword-unit tokenization to create rich token representations. Our evaluation shows that the pNLP-Mixer beats the state-of-the-art of tiny NLP models, pQRRN, and offers sequence tagging performances that are up to 7.8% higher while using *half* of the parameters. The results are remarkable: a pNLP-Mixer model of only *1 million parameters* provides a performance of 99.4% and 97.8% on MTOP and multiATIS, respectively, compared to mBERT which is a a pre-trained model with *170x* more parameters. Our pNLP-Mixer model is simple to implement and accelerate, and provides competitive performance even without pre-training or distillation. Our work demonstrates the importance of projection methods and embedding-free architectures to advance the field of ultra-small models.

References

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

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2017. Quasi-recurrent neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Andrei Z. Broder. 2000. Identifying and filtering near-duplicate documents. In *In COM '00: Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, pages 1–10. Springer-Verlag.

Tom et al Brown. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. 2000. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM transactions on networking*, 8(3):281–293.

Naman Goyal, Jingfei Du, Myle Ott, Giri Anantharaman, and Alexis Conneau. 2021. Larger-scale transformers for multilingual masked language modeling. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*, pages 29–33, Online. Association for Computational Linguistics.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

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

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Prabhu Kaliamoorthi, Sujith Ravi, and Zornitsa Kozareva. 2019. PRADO: Projection attention networks for document classification on-device. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5012–5021, Hong Kong, China. Association for Computational Linguistics.

Prabhu Kaliamoorthi, Aditya Siddhant, Edward Li, and Melvin Johnson. 2021. Distilling large language models into tiny and effective students using pqrnn. *CoRR*, abs/2101.08890.

Guokun Lai, Barlas Oguz, Yiming Yang, and Veselin Stoyanov. 2019. Bridging the domain gap in cross-lingual document classification. *CoRR*, abs/1909.07009.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *Advances in Neural Information Processing Systems (NeurIPS)*.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature*, 521(7553):436.

Haoran Li, Abhinav Arora, Shuohui Chen, Anchit Gupta, Sonal Gupta, and Yashar Mehdad. 2021. Mtop: A comprehensive multilingual task-oriented semantic parsing benchmark. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2950–2962.

Hanxiao Liu, Zihang Dai, David So, and Quoc V Le. 2021. Pay attention to mlps. In *Advances in Neural Information Processing Systems*, volume 34, pages 9204–9215. Curran Associates, Inc.

Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. 2007. Detecting near-duplicates for web crawling. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 141–150.

Patti Price. 1990. Evaluation of spoken language systems: The atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

Sujith Ravi. 2017. Projectionnet: Learning efficient on-device deep networks using neural projections. *CoRR*, abs/1708.00630.

Sujith Ravi. 2019. Efficient on-device models using neural projections. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5370–5379. PMLR.

Sujith Ravi and Zornitsa Kozareva. 2018. Self-governing neural networks for on-device short text classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 887–893, Brussels, Belgium. Association for Computational Linguistics.

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.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *International Conference on Acoustics, Speech and Signal Processing*, pages 5149–5152.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Anshumali Shrivastava and Ping Li. 2014. In defense of minhash over simhash. In *AISTATS*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 886–894. JMLR.org.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, Online. Association for Computational Linguistics.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2022. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6).

Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. 2021. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

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

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*.

Weijia Xu, Batool Haider, and Saab Mansour. 2020. End-to-end slot alignment and recognition for cross-lingual nlu. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5052–5063.

Ping Yu, Mikel Artetxe, Myle Ott, Sam Shleifer, Hongyu Gong, Ves Stoyanov, and Xian Li. 2022. Efficient language modeling with sparse all-mlp.

Sanqiang Zhao, Raghav Gupta, Yang Song, and Denny Zhou. 2021. Extremely small BERT models from mixed-vocabulary training. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2753–2759, Online. Association for Computational Linguistics.