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

Francesco Fusco [1] **Damian Pascual** [1] **Peter Staar** [1]

Abstract

Large pre-trained language models drastically changed the natural language processing (NLP) landscape. Nowadays, they represent the go-to framework to tackle diverse NLP tasks, even with a limited number of annotations. However, using those models in production, either in the cloud or at the edge, remains a challenge due to the memory footprint and/or inference costs. As an alternative, recent work on efficient NLP has shown that small weight-efficient models can reach competitive performance at a fraction of the costs. Here, we introduce pNLP-Mixer, a projection-based MLP-Mixer model for NLP that achieves high weight-efficiency thanks to a novel projection layer. We evaluate our model on two multi-lingual semantic parsing datasets, MTOP and multiATIS. On MTOP our pNLP-Mixer almost matches the performance of mBERT, which has *38 times* more parameters, and outperforms the state-of-the-art of tiny models (pQRNN) with *3 times* fewer parameters. Additionally, on the Hyperpartisan long-sequence classification task our pNLP-Mixer without pretraining outperforms RoBERTa, which has *100 times* more parameters, demonstrating the potential of this architecture.

1. Introduction

Large language models based on the transformer architecture have been fueling the latest successes in Natural Language Processing (NLP). The introduction of these models not only resulted in an improvement of the state-of-the-art, but more importantly, made NLP more accessible by providing a framework that greatly reduces the time needed to solve specific tasks. Self-supervised pretraining is at the heart of this success, as it enables neural models to capture linguistic knowledge from large quantities of readily available data. Once pre-trained, these models can be fine-tuned with a relatively small amount of data to address specific needs. This fine-tuning process is efficient in terms of computational resources and amount of required annotations.

While the merit of large pre-trained language models is undeniable, using models with millions of parameters in production remains a challenge due to the high computational and/or memory requirements. Model size is not only problematic in edge cases, i.e., when deploying on devices with limited memory capacity, but also when running inferences in the server side, since larger models have higher latencies and therefore higher infrastructure costs. There are two main strategies to reduce the inference costs. The first one is to accelerate current model architectures by means of highly optimized hardware in combination with techniques such as pruning (Gordon et al., 2020), quantization (Jacob et al., 2018), and mixed precision (Micikevicius et al., 2018). The second is to use models with radically different architectures, e.g., RNNs, with orders of magnitude fewer parameters that can tackle specific tasks. The rationale behind this second line of research is that not all downstream tasks need the complex linguistic knowledge that transformers contain in their weights, and therefore smaller models may reach competitive performance at a much lower cost. A particularly relevant application of this idea is knowledge distillation into tiny NLP models (Hinton et al., 2015).

Here, we build on this second line of work and we propose *pNLP-Mixer*, a projection-based light-weight MLP-Mixer model for NLP. In computer vision tasks, the MLP-Mixer architecture has been shown to reach near state-of-the-art performance with substantially fewer parameters than the counterparts. MLP-Mixers are appealing models for efficient NLP as: i) unlike RNNs, they operate in parallel, ii) unlike transformers, their complexity scales linearly with the sequence length and yet they can capture long-range dependencies, and, iii) they only contain MLP blocks, for which hardware acceleration is ubiquitous.

Despite all of these desirable characteristics, to the best of our knowledge this is the first time an MLP-mixer model has successfully been applied to NLP tasks. We hypothesize that the reason for this is the difficulty of feeding the model with meaningful features from which it can learn. Indeed, in this work we investigate different embedding-free projection layers and show that the specific projection used does impact downstream performance. This way, our proposed *pNLP-*

[1]IBM Research. Correspondence to: Francesco Fusco <ffu@zurich.ibm.com>.

Mixer is an embedding-free architecture built upon a novel linguistically informed projection layer.

We conduct a thorough investigation of the *pNLP-Mixer* architecture and evaluate different sizes of our model. We compare against pQRNN, a state-of-the-art language model for mobile devices on two semantic parsing datasets, MTOP and multiATIS. On MTOP, our extra-small pNLP-Mixer provides 98.3% of the pQRNN accuracy with *10x* fewer parameters, and the small model achieves better performance with just *one third* of the parameters. Furthermore, our results suggest that this architecture is robust against hyperparameter tuning. Preliminary results also indicate that the validity of the proposed approach is not only confined to datasets for tiny models: on Hyperpartisan, a dataset for long-sequence classification, *pNLP-Mixer* outperforms RoBERTa using *100 times* fewer parameters and no pre-training.

2. Related Work

Since the introduction of transformer-based language models such as BERT (Devlin et al., 2019) or GPT-3 (Brown, 2020) there is a clear trend towards ever larger language models (Brown, 2020; Shoeybi et al., 2019; Goyal et al., 2021; Lample & Conneau, 2019). This steady increase in model size results in unprecedented computational and memory requirements not only at training time, but also at inference time, making the deployment in production of these large models very challenging. Widely used techniques such as quantization (Jacob et al., 2018) and pruning (Gordon et al., 2020) help in reducing the size and latency of existing models. However, with the size of state-of-the-art models growing at an accelerated pace, it is apparent that radically different methods are required. Consequently, efficient NLP has become an important subject both in research and industry (Liu et al., 2021; Menghani, 2021).

In this context, small student models have been shown to approach the performance of larger teacher models when trained via knowledge distillation (Hinton et al., 2015). This is thus one of the most prominent techniques to reduce model size and inference latency. When distilling large language models one of the main questions is which architecture should be used for student models. DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2020), and MobileBERT (Sun et al., 2020) have shown that smaller BERT-like architectures can retain the accuracy of larger BERT-based teacher models. However, while transformer models and attention mechanisms may provide superior performance in many NLP tasks, it is not clear whether they are the most efficient architectural choice for distillation.

Embedding-free models are a family of NLP models designed to avoid storing embedding matrices. Recently, those models have been proposed as students for knowledge distillation (Kaliamoorthi et al., 2021). At their core, they rely on a projection layer that creates, on the fly and without any trainable parameters, input representations from text. Originally, this idea of projection was introduced in ProjectionNet (Ravi, 2017) and (Ravi, 2019) as a mechanism to reduce the model size. In this line, Ravi & Kozareva (2018) presented SGNN, which uses skip-grams and a locality-sensitive hashing function to create dense vectors that are used as model input. Follow up work on projection-based architectures, including SGNN++ (Ravi, 2019), Prado (Kaliamoorthi et al., 2019), pQRNN (Kaliamoorthi et al., 2021) and ProFormer (Sankar et al., 2020) confirms that embedding-free models are a suitable solution for scenarios where model size matters.

Surprisingly, while these works build on the notion of projection, the textual features fed to the locality-sensitive hashing scheme have not received much attention. The input features used in the cited models are n-grams and skip-grams, which can make the projection robust to misspellings (Sankar et al., 2021). However, the features fed to the hashing mechanism can significantly impact the overall model size and performance and so, it deserves deeper investigation. Here, we introduce a new projection layer that can incorporate linguistic knowledge (i.e. morphology) by relying on pre-trained tokenizers, e.g., BERT-tokenizer. In addition, we show that a simple model such as the MLP-mixer (Tolstikhin et al., 2021) can profit from such projections.

3. Model

The pNLP-Mixer has been designed from the ground up as an efficient architecture suitable for both edge cases, i.e., 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. Furthermore, by using only MLPs, this model is not only extremely simple to implement, but also has out-of-the-

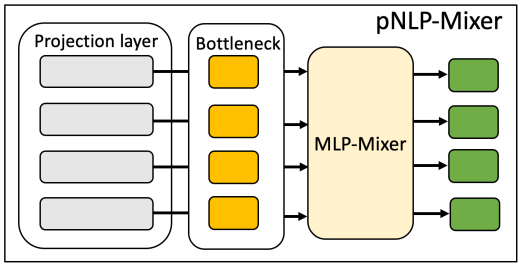

Figure 1. The pNLP-Mixer has a non-trainable projection layer and a linear bottleneck layer followed by a standard MLP-Mixer.

box hardware acceleration in devices ranging from mobile phones to server-grade inferencing accelerators.

The main contribution of this work is to show that a simple model like the MLP-Mixer can represent a valid alternative to transformer-based models in natural language processing, even in a setup where large embedding tables are not used. Not surprisingly, 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 & Motwani, 1998) to create representations from text. While this notion is common to other existing projections, e.g., in pQRNN (Kaliamoorthi et al., 2021), our proposed projection is completely novel. We use MinHash as the hash function given its computational simplicity and we rely on subword tokenization to determine the hash input. Subword tokenization is commonly used in transformers (Sennrich et al., 2016; Schuster & Nakajima, 2012) and it ensures that any string can be represented as a combination of subwords units, i.e., there are no out-of-vocabulary words. In our context, the use of a subword tokenizer provides two main advantages: i) linguistic knowledge can be injected by training a new tokenizer or by using the vocabulary from an available pre-trained language model; ii) the representation of each subword unit can be cached to reduce inference costs since we know beforehand the content of the full vocabulary V, which is usually in the order of 30,000 tokens (Devlin et al., 2019).

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. 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 as follows. In the general case, each element F^u_i, 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^u_i is obtained as the minimum hash value across trigrams, or more formally, $F^u_i = min(h_i(v_0), ..., h_i(v_{k-1}))$. For example, for the subword unit "$Bring$", F^{Bring}_i is computed as $F^{Bring}_i = min(h_i(\text{“}Bri\text{”}), h_i(\text{“}rin\text{”}), h_i(\text{“}ing\text{”}))$, as depicted in Figure 2. 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 subword fingerprint F^u is built by calculating F^u_i for each of the n hash functions $h_0(x)$ to $h_{n-1}(x)$.

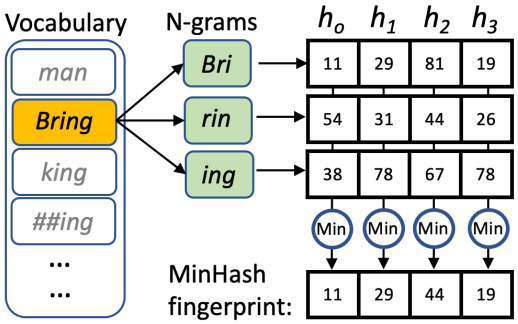

Figure 2. The MinHash fingerprint of a subword unit contains the minimum hash values computed over the trigrams, for each hash function H_{0-3}. Fingerprints for a given token are computed by aggregating the fingerprints of its subword units in a similar way.

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^t_i to the minimum across the j subword fingerprints $F^{u_0}_i$ to $F^{u_{j-1}}_i$. In our example, $F^{Bringing}_i = min(F^{Bring}_i, F^{\#\#ing}_i)$. In practice, this means that by precomputing and caching the fingerprint of each subword unit u in the vocabulary V, inference does not require *any hashing on strings* but only comparisons between integers.

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 of float counters $C^t \in \mathbb{R}^m$ of length m, initially set to zero. In detail, given the fingerprint F^t corresponding to the token t, for each of the n MinHash values F^t_i, we increase by one the value in position p of the array of counters C^t, where $p = F^t_i \ mod \ m$. Therefore, the extracted feature from each token is essentially a Counting Bloom Filter (Fan et al., 2000) storing its MinHash fingerprints. To inject sequence information, we combine single token features by defining a window of size w and concatenating the features from the w tokens to the right and left of the current token, with $w \geq 0$. This way, the input to the model is a matrix C of size $(2w + 1)m \times s$, where s is the maximum sequence length supported by the model, m is the *token feature size* and w the *window size*.

The memory required to enable caching (which is *optional*) is given by an integer matrix storing $|V|$ fingerprints of size n, where $|V|$ is the vocabulary size and n the number of hashes, plus a data structure to map subword units to fingerprints. 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 matrices, which cause major inference bottlenecks as highlighted in previous profiling studies (Iandola et al., 2020), the fingerprints are *not trainable* and they are not directly involved in any matrix multiplication. Cached fingerprints accelerate the computation of *token features*, whose size is configurable and *independent* from the vocabulary size $|V|$. Since fingerprints are not trainable, they can be reused across different models.

In complex NLP pipelines where multiple inferences are required for an individual document, this architecture provides substantial opportunities for optimization. In practice, the features will be extracted exactly once, and multiple pNLP-Mixers will use the same features to address different tasks.

3.2. MLP-Mixer

The MLP-Mixer is a simple architecture that consists exclusively of mixer blocks, where each block has two multilayer perceptrons (MLPs) interleaved by a transposition operation. The transposition of the output of the first MLP lets the second MLP operate on the sequence dimension, effectively mixing information across tokens. Our implementation follows the original work, with skip-connections, layer normalization and GELU non-linearities (Hendrycks & Gimpel, 2016). We refer the reader to Tolstikhin et al. (2021) for a detailed description of this model.

In our case, the matrix $C \in \mathbb{R}^{(2w+1)m \times s}$ produced by the projection layer is passed through a bottleneck layer, i.e., a linear layer, which outputs a matrix $B \in \mathbb{R}^{b \times s}$, where b is the bottleneck size and s the maximum sequence length. This matrix B is the input of the MLP-Mixer model, which in turn produces an output representation $O \in \mathbb{R}^{b \times s}$ of the same dimensions as B. We apply a classification head on top of the output O in order to generate the actual predictions. In the case of semantic parsing this head is a linear layer applied on each token, while for classification tasks we use attention pooling, as in Kaliamoorthi et al. (2021).

4. Experimental Setup

We evaluate our model on two multilingual semantic parsing datasets, multiATIS (Xu et al., 2020) and MTOP. Following Kaliamoorthi et al. (2021) we report slot exact match accuracy for MTOP (Li et al., 2021) and intent accuracy for multiATIS.

MTOP: it covers six different languages, English, Spanish, French, German, Hindi and Thai, and it was created by translating from English to the other languages. The train, validation and test set for each language contain approximately $10,000$ train, $1,500$ validation and $3,000$ test examples. As in (Kaliamoorthi et al., 2021), we evaluate our model on flat (as opposed to hierarchical) slot parsing; the dataset contains 78 different slot labels. The exact match accuracy score is computed as the number of words correctly labeled over the total number of words.

multiATIS: a multilingual version of the ATIS dataset (Price, 1990), which contains queries related to air travel in 9 languages: English, Spanish, French, German, Hindi, Japanese, Portuguese, Turkish and Chinese. For each language except Hindi and Turkish it consists of $4,488$ train, 490 validation and 893 test examples; for Hindi and Turkish the splits are $1,440/160/893$ and $578/60/715$ respectively. We evaluate on intent classification, i.e., determining the intent of a query from 18 different possible labels. The reported metric is intent accuracy, i.e., correctly classified samples over the total number of samples.

5. Model Investigation

Before assessing the final performance our model, we thoroughly investigate the proposed architecture. The experiments in this section are carried out on the validation set of English MTOP and the reported metric is the exact match accuracy of the best epoch. We use as base model a pNLP-Mixer with 2 layers, bottleneck and hidden sizes of 256 and input sequence length of 64. We fix the token feature size to 1024 with window size 1. We train for 80 epochs with learning rate $5e^{-4}$ and batch size 256. We use the vocabulary from BERT base multilingual cased and the fast word piece tokenization algorithm from Song et al. (2021). For a study of the influence of the vocabulary see Appendix A.

5.1. Projection Comparison

First, we compare the impact on performance of different feature extraction strategies, specifically:

BERT Embeddings. We extract the embeddings from the BERT base multilingual cased model. The feature for a token is the average of the embeddings of its subword units.

Binary. We compute n hash functions for each subword unit. Given a token, and a bitmap of size m initialized 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 the bitmap represented as a float tensor.

TSP. We extract a binary feature as in *binary*, but before

Table 1. Comparison of different projection layers for the base model on the validation set of English MTOP.

PROJECTION	EXACT MATCH ACCURACY
BERT	11.0
TSP	77.6
BINARY	79.0
MINHASH	**80.8**
SIMHASH	78.0

feeding it to the model we transform it into a ternary feature, i.e., $\{-1.0, 0.0, +1.0\}$ as described in (Kaliamoorthi et al., 2019).

MinHash. The projection layer described in Section 3.1.

SimHash. We compute the hashes of subword units as in MinHash, but we combine them using SimHash (Manku et al., 2007). 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 respectively a 1 or a 0 in position p.

In Table 1, we report the scores obtained by the base model with each of the projections. The most outstanding result is that the BERT embeddings perform extremely poorly in comparison to the other projections. This is expected since one of the main strengths of BERT is that it produces contextual embeddings, i.e., embeddings that contain information from their surrounding context, and here we need to embed each token in isolation. Regarding the hash-based projections, all of them reach scores in the same range of values. However, there is a considerable difference between the best performing projection, i.e., MinHash, with an exact match accuracy of 80.8%, and the worst one, i.e., TSP, with a score of 77.6%. This difference of over 3% highlights the importance of carefully designing the projection layer and justifies an effort for further research on projection algorithms. Given these results, in the remaining experiments we only consider MinHash as projection layer.

5.2. Model Comparison

The previous result shows that the MinHash projection provides a powerful representation of language. The next question is whether the MLP-Mixer is the optimal architecture to process this representation. To investigate this, we first consider a baseline where the MLP-Mixer is removed and the output of the bottleneck layer is directly passed to the classification heads. Here, we consider two different projection layers, one with window size 1, and another one with window size 4. This way, we question whether the MLP-Mixer uses parameters efficiently or instead, increasing the representational power of the projection would yield better performance. Then, we compare the MLP-Mixer against two other architectures by keeping the same projection, bottleneck layer and classification heads and replacing exclusively the MLP-Mixer with an LSTM (Hochreiter & Schmidhuber, 1997) and a transformer encoder with similar number of parameters[1].

Table 2. Comparison of different architectures with the MinHash projection on the validation set of English MTOP. Projection-only models pass the output of the bottleneck layer directly to the classification head.

MODEL	W. SIZE	# PARAM.	E. MATCH (%)
PROJ.-ONLY	1	820K	65.5
PROJ.-ONLY	4	2.4M	76.5
LSTM	1	1.8M	73.9
TRANSFORMER	1	1.2M	79.4
MLP-MIXER	1	1.2M	**80.8**

In the results from Table 2 we see that simply removing the MLP-Mixer and relying only on the projection produces a significant degradation in performance. In particular, using the projection with window size 1 reduces the number of parameters to 820K but at the cost of a performance decrease of over 15 points. On the other hand, the large projection layer results in a doubling of the number of parameters while reaching only 76.5% exact match accuracy, i.e., 4.3% less than with the MLP-Mixer. From the alternative models, the LSTM performs clearly worse than the MLP-Mixer with a lower exact match accuracy (73.9%) while using 1.8M parameters, i.e., 50% more. The transformer model, with approximately the same number of parameters as the MLP-Mixer (1.2M) reaches a 1.4% lower score. This last result is remarkable: for the same number of parameter the MLP-Mixer outperforms the transformer while having a linear complexity dependency on the input length, rather than quadratic. Overall, this evaluation demonstrates that the MLP-Mixer is a weight-efficient architecture for processing the projection output, i.e., it reaches a higher performance than the alternatives with a smaller amount of parameters.

5.3. Architectural Exploration

Next, we conduct an extensive architectural exploration of the pNLP-Mixer model in order to determine the effect on downstream performance of the different hyperparameters. We distinguish between projection and MLP-Mixer hyperparameters. For the projection, we investigate: *token feature size*, *number of hashes* and *window size*; and for the MLP-Mixer, *bottleneck size* and *number of layers*. As before, we use a learning rate of $5e^{-4}$, batch size of 256 and hidden

[1]We do grid search on the learning rate ($5e^{-4}$, $5e^{-5}$, $1e^{-5}$) and report the best model.

Table 3. Exact match accuracy on the validation set of English MTOP for different configurations of our model. In each column only one element is changed with respect to the base model. We mark in bold the configurations selected for evaluation in the remaining sections.

		PROJECTION						MLP MIXER			
	BASE	1	2	3	4	**5**	6	7	8	9	10
TOKEN FEATURE SIZE	1024	512	2048								
N. HASHES	64			32	128						
WINDOW SIZE	1					0	4				
BOTTLENECK	256							64	512		
N LAYERS	2									1	4
EXACT MATCH (%)	**80.8**	78.4	80.8	78.2	79.9	**80.0**	80.6	77.8	81.0	79.8	80.7
#PARAMETERS	**1.2M**	760K	1.9M	1.2M	1.2M	**630K**	2.7M	340K	2.2M	990K	1.5M

Table 4. Exact match accuracy on the validation set of English MTOP for different configurations of our model. We mark in bold the configurations selected for further evaluation.

		LARGER		SMALLER	
	BASE	**11**	**12**	**13**	14
TOKEN F. SIZE	1024	2048	2048	1024	512
N. HASHES	64	64	64	64	64
WINDOW SIZE	1	1	1	0	0
BOTTLENECK	256	256	512	64	64
N LAYERS	2	4	4	2	2
E. MATCH (%)	80.8	**81.2**	**82.3**	**76.9**	74.5
# PARAM.	1.2M	**2.3M**	**4.4M**	**200K**	180K

size of 256. For each hyperparameter we consider a larger and a smaller value with respect to the base model and in each experiment we change only one hypeparameter in order to isolate its effect. In Table 3 we report exact match accuracy and number of parameters of each configuration.

On the projection side, the token feature size is related to the expressiveness of the projection: larger token features contain more dimensions that can encode information, resulting in fewer collisions and more unique word representations. However, it is also directly proportional to the input dimension of the MLP-Mixer and consequently to the number of model parameters. Interestingly, while reducing the feature size to 512 hurts performance, increasing it to 2048 does not seem to have an impact. Similarly, reducing the number of hashes corresponds to reducing the number of bits used to encode each word, while increasing it over a certain point can result in more collisions, which may harm performance. We observe exactly this behavior with both, the model with 32 hashes and the one with 128 performing worse than the base model. Finally, while a larger window size heavily increases the number of model parameters, it does not produce an improvement in performance. The reason for this is that the mixing of information occurs already at the MLP-Mixer and so, no additional information is added by using a larger window on the projection layer.

Regarding the MLP mixer, increasing the bottleneck size to 512 slightly improves performance while when using 4 layers it reaches a similar value as with 2 layers. However, these hyperparameters are not independent of the projection layer: a larger projection may require a larger MLP-Mixer to process all the information. Therefore, in Table 4 we investigate the relationship between the projection size and the MLP-Mixer. We report results for two larger and two smaller models, and we see that the larger ones, with bigger feature and bottleneck sizes, as well as 4 layers reach the best performance of all the studied models. On the other hand, one of the small models (configuration 13) reaches a 76.9% exact match with only 200K parameters.

In the light of these results, we define four additional models on top of the pNLP-Mixer BASE. A X-LARGE (4.4M) and a LARGE (2.3M) models that correspond to configurations 12 and 11 respectively, as well as a SMALL (630K) and a X-SMALL (200K) models corresponding to configurations 5 and 13.

6. Evaluation

Next, we run a complete evaluation of the five selected configurations of our model on the test sets of MTOP and multiATIS. We train one model per language with identical architecture, hyperparameters and tokenizer (multilingual cased). We compare our pNLP-Mixers against three very large models, i.e., XLU (Lai et al., 2019), which is a large bi-LSTM model with pretrained XLU embeddings; XLM-R (Conneau et al., 2020), a large pretrained multilingual model; and Multilingual BERT (mBERT), released by Devlin et al. (2019); and against two smaller models, pQRNN (Kaliamoorthi et al., 2021) and a simple transformer using the same projection as pQRNN[2].

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

[2]The performance values of these baselines are taken from Kaliamoorthi et al. (2021).

Table 5. Exact match accuracy across languages on the test sets of MTOP. For each column we underline the best overall performance and we mark in **bold** the best performance among the small models: transformer, pQRNNs and pNLP-Mixers.

Model	# Param.	Exact Match Accuracy EN	ES	FR	DE	HI	TH	Avg
XLU	70M (float)	78.2	70.8	68.9	65.1	62.6	68.0	68.9
XLM-R	550M (float)	85.3	81.6	79.4	76.9	76.8	73.8	79.0
mBERT	170M (float)	84.4	81.8	79.7	76.5	73.8	72.0	78.0
Transformer	2M (float)	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
pQRNN distilled	2M (8bit)	81.8	**79.1**	**75.8**	70.8	72.1	69.5	74.8
pNLP-Mixer X-Large	4.4M (8bit)	**83.4**	77.7	74.3	73.5	**74.4**	72.4	**76.0**
pNLP-Mixer Large	2.3M (8bit)	81.9	76.7	74.2	73.0	73.9	**73.9**	75.6
pNLP-Mixer Base	1.2M (8bit)	81.8	75.2	72.7	**73.9**	73.2	72.1	74.8
pNLP-Mixer Small	630K (8bit)	79.9	73.7	70.4	71.2	71.4	70.9	72.9
pNLP-Mixer X-Small	200K (8bit)	77.4	73.3	66.7	69.8	68.4	69.0	70.7

Table 6. Intent accuracy across languages on the test sets of multiATIS. For each column we underline the best overall performance and we mark in **bold** the best performance among the small models: transformer, pQRNN and pNLP-Mixers.

Model	# Param.	Intent Accuracy EN	ES	FR	DE	HI	JA	PT	TR	ZH	Avg
LSTM		96.1	93.0	94.7	94.0	84.5	91.2	92.7	81.1	92.5	91.1
mBERT	170M (float)	95.5	94.1	94.8	95.2	87.8	92.9	94.0	85.4	93.4	92.6
Transformer	2M (float)	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 X-Large	4.4M (8bit)	94.3	93.3	95.7	94.6	86.8	**91.0**	94.2	79.3	92.6	91.3
pNLP-Mixer Large	2.3M (8bit)	95.5	92.8	94.6	94.1	85.0	90.3	95.2	77.8	88.4	90.4
pNLP-Mixer Base	1.2M (8bit)	97.6	91.4	96.1	94.1	87.0	90.5	96.8	83.9	91.6	92.1
pNLP-Mixer Small	630K (8bit)	**98.1**	92.8	93.4	95.1	84.5	90.4	95.4	82.1	94.1	91.8
pNLP-Mixer X-Small	200K (8bit)	94.4	88.2	94.1	95.7	86.8	90.6	89.2	80.0	90.6	90.0

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 versions. We report the performance of pQRNN without distillation as well as with distillation in MTOP, which is a very favorable comparison for pQRNN. In multiATIS pQRNN outperforms mBERT and so, distillation does not provide an advantage.

6.1. MTOP

In Table 5 we see that, as expected, XLM-R and mBERT, which are very large language models, obtain the highest scores. Notably, from the smaller alternatives, our pNLP-Mixer X-Large with only 4,4M parameters and using 8 bit arithmetic, i.e., 150x smaller than mBERT, reaches an average exact match accuracy only 2 and 3 points lower than mBERT and XLM-R respectively. In the same line, our Large model, with a similar size as pQRNN obtains almost 3% higher exact match accuracy across all languages than pQRNN and 0.8% higher than pQRNN with distillation.

From the smaller models, the pNLP-Mixer Base matches the performance of the distilled pQRNN model with half of the parameters, and our Small model outperforms pQRNN by 1% with only 630K parameters, i.e., 3x fewer parameters. Finally, our tiny X-Small model achieves 98.3% of the pQRNN accuracy with *one tenth* of the parameters (200K).

6.2. multiATIS

The evaluation on the multiATIS dataset, reported in Table 6, shows a different trend with respect to MTOP. Here, pQRNN obtains the highest intent accuracy, outperforming even mBERT by 1.8%. From our pNLP-Mixers, we see that a larger size does not correspond to better performance; due to the relatively uniform and simple vocabulary used in ATIS queries, more expressive models are not necessarily better. In fact, our Base model reaches the highest score among the pNLP-Mixers, 92.1%, only a 0.5% less than mBERT with only 1.2M parameters, i.e., 60% of the parameters of pQRNN. The smaller pNLP-Mixer models, Small and X-Small, obtain a competitive performance of 91.8% and 90.0% with a very small fraction of the parameters of any of the alternatives.

It is worth noting, that for the sake of simplicity, we did not tune in any way the pNLP-Mixer models to the multiATIS dataset and so, we rely on the models defined in Section 5.3. It is likely that a slightly higher performance would be obtained with additional tuning to this dataset.

6.3. Discussion

The results presented here show that the pNLP-Mixer constitutes a very performant model for settings where the maximum model size is limited due to either memory or latency requirements. The different models studied in this section highlight the flexibility of this architecture to reduce the number of parameters without a major degradation in performance. In fact, the X-SMALL model with only 200K parameters reaches competitive scores in both datasets, which represents an important step towards ultra-small models for NLP. This evaluation also suggests that the pNLP-Mixer architecture is rather robust against hyperparameter tuning, since the same configuration from MTOP (including the learning rate), performs well in multiATIS.

7. Long-sequence Experiments

As mentioned, a strength of our pNLP-Mixer over attention-based models like transformers is that complexity scales linearly instead of quadratically with the sequence length. Therefore, our model can process long sequences (> 512 tokens) without exceeding the memory requirements of modern hardware, which opens up the possibility of addressing long sequence tasks with a fraction of the compute requirements of current large language models. Although a thorough evaluation of the long-sequence capabilities of our model is beyond the scope of this work, we provide a focused comparison to the Longformer (sequence length 4096) and RoBERTa-base[3]. In particular, we study the two *classification* tasks reported in the evaluation of the Longformer: IMDB and Hyperpartisan. We evaluate our X-LARGE, BASE and X-SMALL models without changing any hyperparameter except for the sequence length.

IMDB is a sentiment classification dataset with two classes, 25,000 train examples and 25,000 test examples. The average sequence length of this dataset is only 300 but over 10% of the examples are longer than 512 tokens. We train our models for 60 epochs, with sequence length 1024. Hyperpartisan is a small dataset (645 samples) with long sequences: the average length is 571 words and 13% of the examples have more than 1,000 words. We train for 60 epochs with sequence length 2048 using the same data split as Beltagy et al. (2020) and report results across five seeds. Note that due to the different sequence length, pNLP-Mixers differ in the number of parameters depending on the task.

[3]We take the performance scores from Beltagy et al. (2020), where the text is chunked to evaluate RoBERTa.

Table 7. Comparison of pNLP-Mixer, RoBERTa and Longformer (seq. length 4096) on the IMDB and Hyperpartisan datasets. The metric in IMDB is accuracy and in Hyperpartisan F1 score.

	IMDB		HYPERPARTISAN	
MODEL	# PARAM.	ACC.	# PARAM.	F1
ROBERTA	125M	95.3	125M	87.4
LONGFORMER	149M	**95.7**	149M	**94.8**
PNLP-MIXER XL	6.3M	82.9	8.4M	89.2
PNLP-MIXER BASE	2.1M	78.6	3.2M	90.6
PNLP-MIXER XS	1.2M	81.9	2.2M	89.8

Table 7 shows that, as expected, in IMDB both, RoBERTa and the Longformer perform clearly better than the pNLP-Mixers, with the Longformer reaching 95.7% accuracy and the best pNLP-Mixer only 82.9%. However, in the Hyperpartisan task, while the Longformer is still the best model, the pNLP-Mixers outperform RoBERTa, with the BASE model reaching 90.6 F1, i.e., 3.2 points more. It is surprising that our tiny pNLP-Mixer models, with up to 120x and 100x fewer parameters than the Longformer and RoBERTa respectively, are competitive in the Hyperpartisan task (even outperforming RoBERTa), without any pretraining nor hyperparameter tuning. The low performance on IMDB, however, calls for a deeper study of this model on complex classification tasks. All in all, this result poses the question of whether larger pNLP-Mixers with pre-training could become a light-weight alternative to large transformer models.

8. Conclusion

In this work we introduce pNLP-Mixer, an embedding-free model that represents the first use of the MLP-Mixer architecture in NLP. We propose a new projection layer to extract features from text and we demonstrate that MLP-Mixers are superior to LSTMs and transformers in processing these features. Our evaluation shows that the pNLP-Mixer matches the performance of the current state-of-the-art of tiny NLP models while being up to *ten times* smaller. Such an all-MLP model is not only simple to implement and to accelerate but also offers linear complexity in the sequence length, which makes it attractive for datacenter use cases. In preliminary results on long sequence classification, we show that, even with *100x* fewer parameters and no pre-training, the pNLP-Mixer outperforms RoBERTa on the Hyperpartisan dataset. This work opens up multiple exciting avenues for future research. Above all, whether the pNLP-Mixers can be enhanced by pre-training, and if so whether larger pre-trained pNLP-Mixers could compete with existing large language models. Moreover, we would also like to investigate the use of this model in distillation setups.

References

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020.

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

Brown, T. e. a. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8440–8451, Online, July 2020. Association for Computational Linguistics.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. 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)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

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

Gordon, M., Duh, K., and Andrews, N. Compressing BERT: Studying the effects of weight pruning on transfer learning. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pp. 143–155, Online, July 2020. Association for Computational Linguistics.

Goyal, N., Du, J., Ott, M., Anantharaman, G., and Conneau, A. Larger-scale transformers for multilingual masked language modeling. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*, pp. 29–33, Online, August 2021. Association for Computational Linguistics.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

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

Iandola, F., Shaw, A., Krishna, R., and Keutzer, K. SqueezeBERT: What can computer vision teach NLP about efficient neural networks? In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pp. 124–135, Online, November 2020. Association for Computational Linguistics.

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

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. 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)*, June 2018.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4163–4174, Online, November 2020. Association for Computational Linguistics.

Kaliamoorthi, P., Ravi, S., and Kozareva, Z. 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)*, pp. 5012–5021, Hong Kong, China, November 2019. Association for Computational Linguistics.

Kaliamoorthi, P., Siddhant, A., Li, E., and Johnson, M. Distilling large language models into tiny and effective students using pqrnn. *CoRR*, abs/2101.08890, 2021.

Lai, G., Oguz, B., Yang, Y., and Stoyanov, V. Bridging the domain gap in cross-lingual document classification. *CoRR*, abs/1909.07009, 2019.

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

Li, H., Arora, A., Chen, S., Gupta, A., Gupta, S., and Mehdad, Y. 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*, pp. 2950–2962, 2021.

Liu, X., Sun, T., He, J., Wu, L., Zhang, X., Jiang, H., Cao, Z., Huang, X., and Qiu, X. Towards efficient NLP: A standard evaluation and A strong baseline. *CoRR*, abs/2110.07038, 2021.

Manku, G. S., Jain, A., and Sarma, A. D. Detecting near-duplicates for web crawling. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pp. 141–150, 2007.

Menghani, G. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *CoRR*, abs/2106.08962, 2021.

Micikevicius, P., Narang, S., Alben, J., Diamos, G. F., Elsen, E., García, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed precision training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Price, P. 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*, 1990.

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

Ravi, S. Efficient on-device models using neural projections. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5370–5379. PMLR, 09–15 Jun 2019.

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

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

Sankar, C., Ravi, S., and Kozareva, Z. Proformer: Towards on-device LSH projection based transformers. *CoRR*, abs/2004.05801, 2020.

Sankar, C., Ravi, S., and Kozareva, Z. On-device text representations robust to misspellings via projections. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 2871–2876, Online, April 2021. Association for Computational Linguistics.

Schuster, M. and Nakajima, K. Japanese and korean voice search. In *International Conference on Acoustics, Speech and Signal Processing*, pp. 5149–5152, 2012.

Sennrich, R., Haddow, B., and Birch, A. 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)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019.

Song, X., Salcianu, A., Song, Y., Dopson, D., and Zhou, D. Fast wordpiece tokenization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 2089–2103, 2021.

Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., and Zhou, D. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2158–2170, Online, July 2020. Association for Computational Linguistics.

Tolstikhin, I., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., Lucic, M., and Dosovitskiy, A. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.

Xu, W., Haider, B., and Mansour, S. 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)*, pp. 5052–5063, 2020.

A. Vocabulary Comparison

Here, we evaluate the influence of the vocabulary used in the tokenization step. This way, we aim at determining whether accurate tokenization is needed or the mixing of information between subtokens that occurs inside the MLP-Mixer model compensates for a poor tokenization. We evaluate our BASE model with the MinHash projection layer using the BPE tokenization algorithm, i.e., the same as used by BERT, and different vocabularies. In particular, we compare the base multilingual cased vocabulary (M) with a vocabulary from each of the languages in the MTOP dataset, i.e., English (EN), Spanish (ES), French (FR), German (DE), Hindi (HI) and Thai (TH)[4]. In Table 8, we report exact match accuracy on the development set for each of the language-specific datasets in MTOP.

Table 8. Exact match accuracy across languages on the validation sets of MTOP for different vocabularies.

VOC.	EN	ES	FR	DE	HI	TH	AVG
EN	79.0	74.3	72.1	73.2	51.8	5.9	59.4
ES	81.4	75.4	72.4	72.3	5.3	6.1	52.2
FR	**81.6**	**75.8**	**73.6**	72.3	5.3	6.1	52.5
DE	80.3	70.7	68.9	72.0	6.0	5.8	50.6
HI	81.3	75.2	72.8	**73.3**	68.6	45.1	69.4
TH	48.1	40.6	41.8	26.9	7.8	58.9	37.4
M	80.8	73.7	72.4	71.5	**69.5**	**68.6**	**72.8**

The results show that using a vocabulary from either of the European languages (English, Spanish, French and German) has only a minor influence on performance for the datasets in one of these languages. However, when evaluated on Hindi and Thai, a model using a vocabulary from a European language performs very poorly, with exact match accuracy values around 5% except for English on Hindi (51.8%). Surprisingly, the model with Hindi vocabulary reaches competitive performance across all languages except for Thai. It is worth noting that the Hindi vocabulary contains a very large amount of English words, what may explain the similar performance to the English vocabulary. On the other hand, the Thai vocabulary performs poorly on all languages except for Thai itself (58.9%), where it still lags behind the multilingual vocabulary (68.6%). Indeed, the multilingual vocabulary reaches the best overall performance, with the evaluation on European languages and Hindi reaching very similar scores to the corresponding language-specific vocabularies, and clearly outperforming the Thai vocabulary on the Thai dataset.

While these results suggest that tokenization is an important step of our model, they also show that there is not a clear gain in using language-specific vocabularies over the multilingual vocabulary.

[4]These are available at https://huggingface.co