Adapting XLM-RoBERTa for Ancient and Historical Languages: A Resource-Efficient Approach to Morphological Analysis and Gap-Filling

Aleksei Dorkin
Institute of Computer Science
University of Tartu
aleksei.dorkin@ut.ee

Kairit Sirts
Institute of Computer Science
University of Tartu
kairit.sirts@ut.ee

Abstract

We present our submission to the uncostrained subtask of the SIGTYP 2024 Shared Task on Word Embedding Evaluation for Ancient and Historical Languages for morphological annotation, POS-tagging, lemmatization, character- and word-level gap-filling. We developed a simple, uniform, and computationally lightweight approach based on the adapters framework using parameter-efficient fine-tuning. We applied the same adapter-based approach uniformly to all tasks and 16 languages by fine-tuning stacked language- and task-specific adapters. Our submission obtained an overall second place out of three submissions, with the first place in word-level gap-filling. Our results show the feasibility of adapting language models pre-trained on modern languages to historical and ancient languages via adapter training.

1 Introduction

The application of natural language processing techniques and pre-trained language models to analysis of ancient and historical languages is a compelling subject of research that has been so far overlooked. While there exist a number of benchmarks, such as GLUE (Wang et al., 2018), SuperGLUE (Wang et al., 2019), or XGLUE (Liang et al., 2020), for evaluating the quality of embeddings and language models for regular, modern language, such benchmarks are lacking for ancient and historical languages. Thus, the SIGTYPE 2024 Shared Task on Word Embedding Evaluation for Ancient and Historical Languages shared task contribute to cover this gap.

In the current transformer-based language models paradigm, the most common approach to train a model for a particular task in a particular language is to fine-tune a multilingual or language-specific pre-trained model on the target task data. However, large pre-trained language models are predominantly trained on corpora of modern languages, with few exceptions such as LatinBERT (Bamman and Burns, 2020). Ancient and historical languages generally lack sufficient data to perform full pretraining of large language models or to continue training from a checkpoint trained on some different language. Full fine-tuning of modern language models on a relatively small amount data in an ancient/historical language might lead to overfitting and catastrophic forgetting. These considerations are also common in context of other low resource tasks or domains.

Several approaches have been proposed to alleviate these issues. For instance, the supplementary training approach proposed by Phang et al. (2018) involves first fine-tuning a pretrained model on an intermediary labeled task with abundant data, and then on the target task which may have limited data. This approach showed gains over simply fine-tuning on the target task. Pfeiffer et al. (2020b) developed a cross-lingual transfer-learning approach based on the adapters framework for parameter efficient fine-tuning of language models (Houlsby et al., 2019; Bapna and Firat, 2019). Their approach involves fine-tuning a language adapter and a task adapter stacked on top of each other. This adapter-based method is expanded by Pfeiffer et al. (2020c) by adopting a custom tokenizer and an embedding layer. Several ways of initializing the new embedding layers are compared on underrepresented modern languages. Since ancient and historical languages are underrepresented, the same techniques should be applicable in this context as well.

A useful feature of both the supplementary training approach of Phang et al. (2018) and the adapter-based training of Pfeiffer et al. (2020b) is that they provide a uniform framework that can be applied to different languages and tasks in a similar manner. While previous related works mainly focused on modern languages, we aim to assess the feasibility of this unified approach for ancient and historical

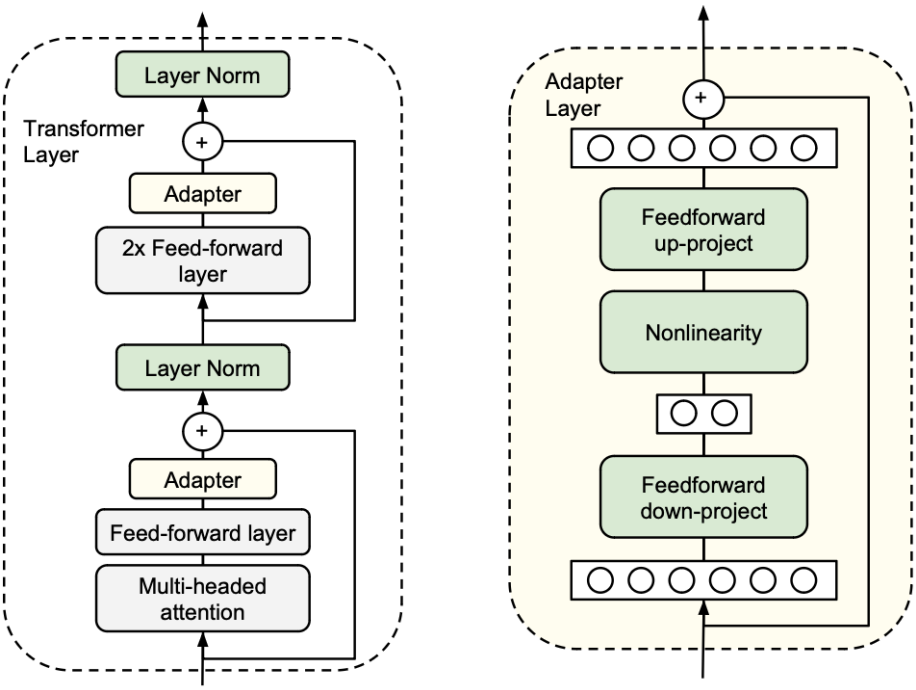

Figure 1: An illustration of the Bottleneck Adapter from (Houlsby et al., 2019). The left side demonstrates how a bottleneck adapter is added to a single transformer layer, while the structure of an individual adapter layer is on the right. Only elements in green are trained, while the rest remains frozen.

languages.

Our submission to the SIGTYP 2024 Shared Task on Ancient on Word Embedding Evaluation for Ancient and Historical Languages adopts the methods described by Pfeiffer et al. (2020b) and Pfeiffer et al. (2020c) by stacking fine-tuned language and task adapters, and customizing the tokenizer and the embedding layers. Our system is implemented as a unified framework, where various pre-trained models, languages, and tasks can be plugged in. The system was evaluated on POS tagging, morphological annotation, lemmatization, and filling-in both word-level and character-level gaps for 16 ancient and historical languages. We participated in the unconstrained subtask, which allowed using any additional resources, such as pre-trained language models. Out of 3 participants on the leaderboard we took a close second place, with the first place on the word gap-filling task. Our main contribution is showing that by adopting the parameter-efficient adapter training methodology, the large language models pre-trained on modern languages are applicable also to low-resource ancient and historical languages.

2 Adapters

There exists a number of approaches to parameter efficient fine-tuning. In our submission to the shared task the focus is on adapters exclusively.

Houlsby et al. (2019) propose a parameter efficient fine-tuning strategy that involves injection of a number of additional trainable layers into the original architecture. The architecture of the proposed strategy is illustrated on Figure 1. In each transformer layer an adapter layer is added twice: after the attention sub-layer and after the feedforward sub-layer. To limit the number of trainable parameters, the authors propose a bottleneck architecture of adapter layers: the adapter first projects the input into a smaller dimension, applies non-linearity, then projects back to the original dimension. This is known as the bottleneck adapter.

Pfeiffer et al. (2020b) extend on the strategy in context of cross-lingual transfer learning in two ways. First, the adapter stack technique is introduced (illustrated on Figure 2), which is primarily used to separate language adaptation training and task-specific training. Meaning that first a language adapter is trained, then a task-specific adapter is stacked on top of it and trained (while the language adapter is frozen). Secondly, the bottleneck adapter is expanded upon to include the embedding layer

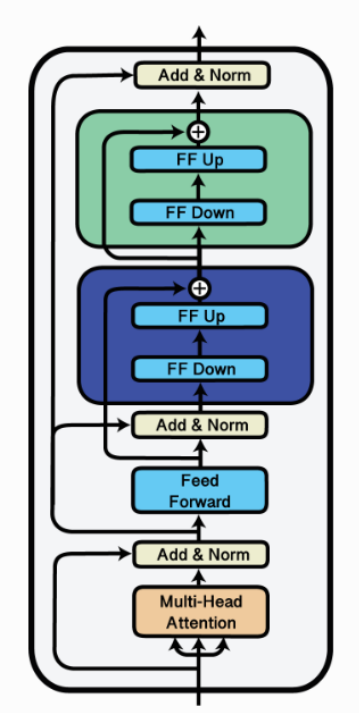

Figure 2: An illustration of an adapter stack as presented on the AdapterHub documentation page[1]. Blue and green blocks represent different adapter layers stacked on top of each other.

as well—the invertible adapter is introduced that transforms both input and output token representations to further improve language adaptation.

3 Methodology

Our approach in this work is based on training language adapters and task specific adapters for XLM-RoBERTa (Conneau et al., 2020)—the multilingual variant of RoBERTa (Liu et al., 2019) trained on 100 different languages. Since many of the languages in the shared task have related modern languages in the training data of XLM-RoBERTa, or are even included themselves (such as Latin), we expect to benefit from knowledge transfer.

3.1 Data

The dataset provided by the organizers is a compilation of various resources (Bauer et al., 2017; Doyle, 2018; Ó Corráin et al., 1997; HAS Research Institute for Linguistics, 2018; Zeman et al., 2023; Acadamh Ríoga na hÉireann, 2017; Simon, 2014) and comprises 16 languages in total spanning several historical epochs and upper bounded by 1700 CE. The information on the languages in the dataset is presented in Table 1.

3.2 Adapter Training

Full fine-tuning of language models has several disadvantages such as the possibility of catastrophic forgetting and the necessity to train and maintain a full copy of the model weights for each individual task. When applying the technique first proposed in (Phang et al., 2018) in a multi-lingual and multi-task setting, the scale of the mentioned disadvantages is magnified. Training and maintaining a separate copy of the model for each combination of language, target task, and intermediate task can become quite onerous. Meanwhile, each copy of the model is narrowly specialized in a single task, with generalization capabilities being potentially limited. This highlights the practicality of parameter efficient fine-tuning in comparison.

The overall approach is, in general, the same for every language. First, we train a language adapter for every language individually. A language adapter is comprised of a bottleneck adapter and masked language modeling prediction head, the training objective is, accordingly, masked language modeling. This is based on the assumption that a significant part of the model's parameters is not actually language-specific, and thus does not need changing. We simply need to adapt the model to a new language. The language adapter is trained for 10 epochs regardless of the size of the data. Unmasked part of the training data for the word-filling task is used, while the masked part is not utilized for training. Secondly, we train task-specific adapters for each language. In this setup we use the adapter stack: the language adapter is loaded, but frozen, and we add a task-specific adapter on top of it, and train said adapter. We have two tasks per language: morphological tagging, which combines both POS-tagging and morphological annotation, and lemmatization, which is also implemented as sequence tagging problem. Similarly to language modeling, in both tasks the adapters are trained for 10 epochs.

3.3 Custom tokenizers and embeddings

Some of the languages in the shared task are not covered by the model's tokenizer. In other words, a significant portion of the tokenizer's output includes <unk> tokens. For each of these languages a new tokenizer is initialized. In addition to that, a

[1] https://docs.adapterhub.ml/adapter_composition.html

Language	Code	Train Sentences	Valid Sentences	Test Sentences
Ancient Greek	grc	24,800	3,100	3,101
Ancient Hebrew	hbo	1,263	158	158
Classical Chinese	lzh	68,991	8,624	8,624
Coptic	cop	1,730	216	217
Gothic	got	4,320	540	541
Medieval Icelandic	isl	21,820	2,728	2,728
Classical and Late Latin	lat	16,769	2,096	2,097
Medieval Latin	latm	30,176	3,772	3,773
Old Church Slavonic	chu	18,102	2,263	2,263
Old East Slavic	orv	24,788	3,098	3,099
Old French	fro	3,113	389	390
Vedic Sanskrit	san	3,197	400	400
Old Hungarian	ohu	21,346	2,668	2,669
Old Irish	sga	8,748	1,093	1,094
Middle Irish	mga	14,308	1,789	1,789
Early Modern Irish	ghc	24,440	3,055	3,056

Table 1: Language statistics in the Shared Task.

new embedding layer is also initiliazed. The initial weights are copied from the original embeddings for tokens that overlap with the multilingual tokenizer.

To decide whether the model requires a custom tokenizer, two checks are made:

1. The percentage of unknown tokens > 5%
2. <unk> token is in top 10 by frequency

If either of these conditions is true, a custom tokenizer is created. Refer to Table 2 for detailed statistics. Based on these conditions, 5 languages need a separate tokenizer: Old Church Slavonic, Coptic, Classical Chinese, Old Hungarian, and Old East Slavic. When selecting the vocabulary size, the aim was to have as many full words as possible in the vocabulary, but at the same time allow for morphological variation. Thus, the number of items in the vocabulary should be neither too high nor too low. For that reason, we selected the size to be 3000 for all languages, except Classical Chinese. For Classical Chinese the size was insufficient, so it was increased to 10000.

Pfeiffer et al. (2020c) suggest several possible options of initializing the weights of the custom embedding layer corresponding to the new tokenizer. These include random initialization, copying the weights of tokens that overlap between multilingual and language-specific tokenizers, as well as a matrix factorization-based approach that aims to identify latent semantic concepts in the original embedding matrix that are useful for transfer. It is noted, however, that the latter approach shows less gains when used with languages with underrepresented scripts. For this reason and due to simplicity, with settle with the lexical overlap approach for all affected languages.

lang	# tokens	% unknown	<unk> rank
chu	495612	15.31%	1
cop	39771	45.61%	2
fro	55448	0.00%	-
ghc	1282852	0.00%	-
got	98874	1.05%	-
grc	1079457	0.47%	-
hbo	124063	0.36%	-
isl	688580	0.00%	-
lat	296770	0.00%	-
latm	897209	0.00%	-
lzh	408259	1.40%	4
mga	492894	0.00%	-
ohu	352811	5.05%	1
orv	592890	5.00%	2
san	59349	0.01%	-
san	190711	0.00%	-

Table 2: Tokenizer coverage statistics. <unk> ranks not present in top 10 are not reported.

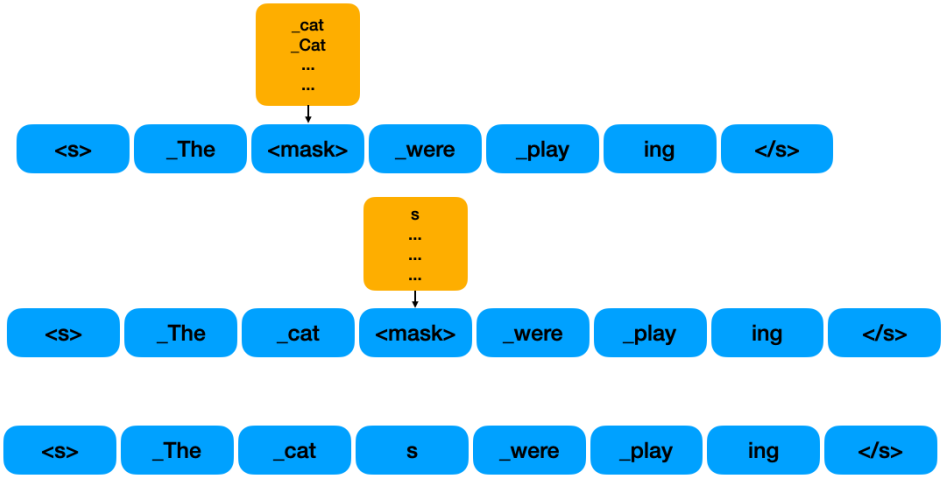

Figure 3: A schematic illustration of the decoding process for word-level mask filling. Blue boxes represent current tokens in the sentence, while orange boxes represent the probability distribution of tokens at the position of the mask token. The upper half represents the first step of decoding. We start with predicting the most likely replacement for the leftmost masked token that starts with _ symbol representing the beginning of a new word. Then, we replace the mask with that token and append a new mask token to the right of it, as represented in the middle part. We predict the most likely replacement for the new mask token. If it starts with _ symbol, we discard the mask token, consider the word predicted, and move to the next masked word if it's present. Conversely, if there's no _ in the predicted token, we append it to the previously predicted token. We repeat this process k times, or until we encounter a token starting with _. k is a hyperparameter that may be tuned, however increasing k increases the decoding time significantly. For this reason we set k to 1 for all languages. At the bottom of the figure the final result with no mask tokens is demonstrated. Note that this description is specific to XLM-RoBERTa tokenizer, other model's tokenizers may have different behaviour.

3.4 Tasks

Filling Word-level Gaps Having trained a masked language modeling adapter for each language, we already have a suitable model to perform word-level gap filling. However, the fact XLM-RoBERTa, as well as most modern language models, is based on subword tokens rather than full words becomes a significant hurdle. The reason is the we can not know in advance whether the target word we want to predict is comprised of one or more subword units, and that makes the decoding more complicated. To address this issue, for each sentence with masked words in it, we follow the steps (see Figure 3):

1. Locate the first <mask> token
2. Use the model's prediction to determine the most likely token that starts with the whitespace prefix
3. Replace the <mask> token with the prediction, and look ahead up to k steps: does the next token start with the whitespace token? If it does, we break the cycle, and continue processing the remaining <mask> tokens in the sentence, otherwise we append the next predicted token to our previous prediction, and repeat the look ahead.

For Classical Chinese, the process is simplified, and we simply predicted the most likely token.

Filling Character-level Gaps The approach to character-level mask filling is purely algorithmic. We consider a sequence of characters of with no whitespaces to be a masked word. A dictionary is built based on training data. For each masked word a look up among all the words of the same length is performed: a regular expression built from the masked word by replacing "[_]" with "." is matched against these words. If there are matches, up to the first 3 matches are returned. For each match character replacements are extracted from each candidate. If there are no matches, and the masked word contains only one "[_]", the masked word is split in two parts, and each part is matched against the dictionary. If there is at least one match, the whitespace is returned as the replacement.

The described algorithmic method could be extended in two ways:

1. Multiple candidates could be reranked using the trained language model

2. If there are no candidates, a suitable word could be generated using the trained model, similarly to the word-level gap-filling task

However, the extensions were not developed for the shared task due to the following. Each sentence may contain multiple masked words, and in addition the whitespace symbol may be masked as well. This significantly increases the solution's complexity, yet the magnitude of the benefit is unclear.

Finally, for Classical Chinese the following approach was applied. First, the symbols in the original dataset were decomposed into the lowest possible forms which are mostly strokes and small indivisible units using the Hanzipy library[2], then a masked language model adapter was trained on that decomposed data. The model achieved relatively high masked language modeling evaluation accuracy—about 40%—on the validation set, as measured the by the Trainer class in the transformers library. The trained model was then used to predict the most likely replacement for each masked symbol. The ground truth data provided by the organizers, however, remained in the original composed form. This proved to be a significant problem. An attempt was made to compose the symbols back into their original form algorithmically, since the library used for decomposition doesn't have the option to reconstruct the original symbol from its constituents, to our best understanding. However, this attempt was unsuccessful given that our submission attained 0 score on the test set.

POS-tagging & Full Morphological Annotation We frame both POS-tagging and full morphological annotation as a single token classification task. For each token we collect all available morphological features and the POS tag from the annotation. Then we concatenate them as a single string, which is then used as the class to predict. Then we employ the adapter stack technique: we load the corresponding language adapter, and create a task specific adapter with token classification prediction head. We only train the task specific adapter. For inference we decompose the predicted class strings back into individual tags. The approach is the same for every language.

[2] https://github.com/Synkied/hanzipy

Lemmatization Similarly to the previous task, we frame lemmatization as token classification task. To achieve that we generate transformation rules for each lemma/form pair based on the technique first introduced in (Straka, 2018) and expanded upon in (Dorkin and Sirts, 2023). The rules are comprised of individual edits that need to be made to transform the given form into its lemma. The edits are represented as a single string. We use that string as the label to predict. Consequently, inference is done in two steps: first the rule for each token is predicted, then that rule is applied to the form and the resulting lemma is returned. The approach is the same for all language except for Classical Chinese. For Classical Chinese we do not train a model, but rather use a simple dictionary look up which results in nearly 100% accuracy.

3.5 Technical Implementation Details

The implementation is primarily based on the Adapters[3] library and the provided training script examples. The most significant change in the training scripts is the addition of custom embedding layer initialization. Some aspects of lemmatization as token classification task were adopted from our previous work. For languages not utilizing custom tokenizers we employ the invertible bottleneck adapters (Pfeiffer et al., 2020b), while for the languages that do require custom tokenizers the employ the regular bottleneck adapters (Houlsby et al., 2019). The motivation is that when we have a custom embedding layer, there is no need to adapt it additionally. In all experiments we use the base version of XLM-RoBERTa due to time constraints, however we can well expect that the large version would show improved performance. The code and the instructions to reproduce are available on the project's GitHub repository[4].

To train a new tokenizer that remains compatible with the original model (i.e. retains all the special tokens and their indices) we use `train_new_from_iterator` method of the Tokenizer class instance in HuggingFace transformers associated with model and supply it with same data we use for masked language modeling training.

For training we used a single NVIDIA Tesla V100 GPU on the University's High Performance Cluster (University of Tartu, 2018). The training time for a single task (including masked language modeling) on average was about 10-20 minutes,

[3] https://github.com/adapter-hub/adapters
[4] to-be-added-in-camera-ready-version

	Overall results	POS-tagging	Lemmatization	Morphological analysis	Character gap-filling	Word gap-filling
Ancient Greek	0.70	0.96	0.94	0.97	0.61	0.03
Ancient Hebrew	0.61	0.94	0.97	0.95	0.19	0.00
Classical Chinese	0.57	0.83	1.00	0.89	0.00	0.10
Coptic	0.52	0.61	0.75	0.75	0.45	0.02
Gothic	0.70	0.93	0.93	0.92	0.67	0.03
Medieval Icelandic	0.73	0.97	0.98	0.96	0.57	0.17
Classical and Late Latin	0.73	0.96	0.97	0.96	0.66	0.11
Medieval Latin	0.76	0.99	0.99	0.99	0.70	0.14
Old Church Slavonic	0.50	0.66	0.60	0.67	0.54	0.02
Old East Slavic	0.56	0.76	0.69	0.80	0.48	0.06
Old French	0.69	0.95	0.92	0.98	0.52	0.07
Vedic Sanskrit	0.66	0.84	0.88	0.86	0.65	0.05
Old Hungarian	0.52	0.75	0.63	0.76	0.46	0.00
Old Irish	0.19	-	-	-	0.35	0.03
Middle Irish	0.22	-	-	-	0.39	0.04
Early Modern Irish	0.28	-	-	-	0.50	0.06

Table 3: Results of our submission on the leadearboard. For the variants of Irish, training and test data was only provided for gap-filling tasks.

ranging from 2 to 40 minutes depending on the amount of data, which is notably less than it would take for full fine-tuning.

Due to an error that was present in our code, initially our models did not use a newly trained masked language modeling prediction head weights, but rather relied on the weights of the original XLM-RoBERTa's weights. To our surprise, after fixing the error, the difference turned out to be quite negligible for languages that did not rely on custom embeddings.

4 Results

In the results on Table 3 we see that our approach seems to generally underperform on languages that require custom tokenizers and embeddings: Classical Chinese, Coptic, Old Church Slavonic, Old East Slavic, Old Hungarian. This can be explained by the amount of data being quite limited to be able to train meaningful input representations, while the lexical overlap technique failed to provide significant benefit due to significant differences in their scripts. We hypothesize that a more sophisticated approach to embedding initialization and tokenizer training could result in considerable benefits.

We note a surprisingly strong performance of our algorithmic approach to character-level gap-filling with the exception of Classical Chinese where the approach is simply not applicable. We expect the extensions to the approach outlined in Section 3.4 could improve the results further, however we do not expect substantial gains compared to the required effort due to the way the task is formalized. Due to the whitespaces appearing as masked characters there is no way to know in advance how many individual words in the sentence in total have masked characters in them. In addition to that, a single word may contain several masked characters. The combination of these two factors makes improvements based on language model rescoring quite computationally expensive. For Classical Chinese, however, we hypothesise that the score would be considerably higher if the ground truth was also in the decomposed form, because to the best of our understanding, reconstructing the original characters from individual strokes is not a straightforward task.

Despite taking the first place in the word-level gap filling task, the absolute scores are still very low. This is to be expected, because the task is compounded by the presence of multiple masked words per sentence, as well the reliance on subword tokenization of our approach. Additionally,

the implementation may not be suitable at all for languages due to differences in the writing system. Finally, we surmise that due to the amount of training being quite small for masked language modeling problem, we can not expect to be able to correctly predict completely new words, however we can, in theory, predict previously unseen word forms due to subword tokenization. This means that experimenting with the k parameter used to limit the number of times the look ahead for word continuation is performed in the word-level gap filling problem could improve the results somewhat.

The pattern-based approach to lemmatization performs also reasonably well in most languages. The pattern-based approach to lemmatization is restricted by the set of patterns derived based on the training set. If the evaluation set contains many words for which a suitable pattern is missing, the system simply cannot make a correct prediction. However, we believe that this is not the problem here. In particular, we note that those languages who have lower scores on lemmatization, also perform consistently lower on other tasks. Thus, we suggest that the lower performance is rather related to insufficient data for training meaningful representations. In addition to underpeforming on languages with underrepresented writing systems, we observe somewhat low performance on Vedic Sanskrit as well. One explanation could be that the usage of diacritics may negatively affect the character-based transformation rule generation; however this hypothesis requires further investigation.

Finally, our combined approach to POS-tagging and morphological analysis performs quite well on most of the languages. This is somewhat unexpected, because the model is limited in predicting only the combinations of part-of-speech and morphological features that are present in training data.

5 Conclusion

This paper described our solution to the SIGTYP 2024 Shared Task developed based on the adapters framework. The system is simple and uniform, and can be easily extended to other tasks and languages. For each language we trained a language adapter, and two task adapters, one for POS and morphological tagging and one for lemmatization. For languages with scripts underrepresented the XLM-RoBERTa base model vocabulary, we additionally created custom tokenizers and embeddings. One notable advantage of the adopted approach is its resource efficiency—adapting the model to a new language and task can be done in less than an hour. As a future work, the system could be improved with more advanced adapter-based techniques such as adapter fusion (Pfeiffer et al., 2020a) to leverage typological relatedness of languages.

References

Acadamh Ríoga na hÉireann. 2017. Corpas Stairiúil na Gaeilge 1600-1926. Retrieved: June 10, 2022.

David Bamman and Patrick J. Burns. 2020. Latin bert: A contextual language model for classical philology.

Ankur Bapna and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. 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 1538–1548.

Bernhard Bauer, Rijcklof Hofman, and Pádraic Moran. 2017. St. Gall Priscian Glosses, version 2.0.

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.

Aleksei Dorkin and Kairit Sirts. 2023. Comparison of current approaches to lemmatization: A case study in estonian. In *Proceedings of the 24th Nordic Conference on Computational Linguistics (NoDaLiDa)*, pages 280–285.

Adrian Doyle. 2018. Würzburg Irish Glosses.

HAS Research Institute for Linguistics. 2018. Old Hungarian Codices.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Yaobo Liang, Nan Duan, Yeyun Gong, Ning Wu, Fenfei Guo, Weizhen Qi, Ming Gong, Linjun Shou, Daxin Jiang, Guihong Cao, et al. 2020. Xglue: A new benchmark dataset for cross-lingual pretraining, understanding and generation. *arXiv preprint arXiv:2004.01401*.

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

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020a. Adapterfusion: Non-destructive task composition for transfer learning. *CoRR*, abs/2005.00247.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020b. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. *arXiv preprint arXiv:2005.00052*.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020c. Unks everywhere: Adapting multilingual language models to new scripts. *arXiv preprint arXiv:2012.15562*.

Jason Phang, Thibault Févry, and Samuel R Bowman. 2018. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*.

Eszter Simon. 2014. Corpus Building from Old Hungarian Codices. In Katalin É. Kiss, editor, *The Evolution of Functional Left Peripheries in Hungarian Syntax*. Oxford University Press, Oxford.

Milan Straka. 2018. UDPipe 2.0 prototype at CoNLL 2018 UD shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 197–207, Brussels, Belgium. Association for Computational Linguistics.

University of Tartu. 2018. Ut rocket.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Daniel Zeman, Joakim Nivre, Mitchell Abrams, Elia Ackermann, et al. 2023. Universal dependencies 2.12.

Donnchadh Ó Corráin, Hiram Morgan, Beatrix Färber, Benjamin Hazard, Emer Purcell, Caoimhín Ó Dónaill, Hilary Lavelle, Julianne Nyhan, and Emma McCarthy. 1997. CELT: Corpus of Electronic Texts. Retrieved: March 15, 2021.