Mini-Model Adaptation: Efficiently Extending Pretrained Models to New Languages via Aligned Shallow Training

Kelly Marchisio[1,2] * **Patrick Lewis**[2] † **Yihong Chen**[3,4] **Mikel Artetxe**[5] †
[1]Johns Hopkins University [2]Cohere AI [3]Meta AI
[4]University College London [5]Reka AI
kmarc@jhu.edu, mikel@reka.ai

Abstract

Prior work shows that it is possible to expand pretrained Masked Language Models (MLMs) to new languages by learning a new set of embeddings, while keeping the transformer body frozen. Despite learning a small subset of parameters, this approach is not compute-efficient, as training the new embeddings requires a full forward and backward pass over the entire model. We propose *mini-model adaptation*, a compute-efficient alternative that builds a shallow *mini-model* from a fraction of a large model's parameters. New language-specific embeddings can then be efficiently trained over the mini-model and plugged into the aligned large model for rapid cross-lingual transfer. We explore two approaches to learn mini-models: MINIJOINT, which jointly pretrains the primary model and the mini-model using a single transformer with a secondary MLM head at a middle layer; and MINIPOST, where we start from a regular pretrained model, build a mini-model by extracting and freezing a few layers, and learn a small number of parameters on top. Experiments on XNLI, MLQA and PAWS-X show that mini-model adaptation matches the performance of the standard approach using 2.3x less compute on average.

1 Introduction

Recent work on multilingual NLP has focused on pretraining (masked) language models on unlabeled corpora in multiple languages (Pires et al., 2019; Conneau et al., 2020; Xue et al., 2021). The resulting models can then be finetuned using labeled downstream data in a single language (typically English), and zero-shot transferred to the rest of the languages. While effective, existing models rarely cover more than a few dozen languages, and pretraining new models from scratch to support additional languages can be prohibitively expensive.

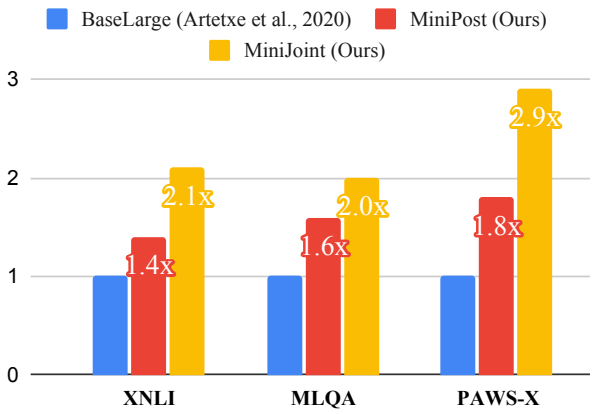


Figure 1: **Average speedup of mini-model adaptation over Artetxe et al. (2020)**. A speedup of x means that our approach needs x times less compute to achieve the same performance. See §4.2 for more details.

Motivated by this, a recent line of work has explored pretraining an initial model in a few languages, and expanding it to new languages post-hoc in a continual learning fashion (M'hamdi et al., 2022). More concretely, Artetxe et al. (2020) showed that it is possible to expand an English masked language model (MLM) to new languages by freezing the transformer body and learning a new embedding layer using the original MLM objective. Recent work has reported improved results by using a better initialization scheme (Pfeiffer et al., 2021), or learning additional language-specific parameters through adapters (Pfeiffer et al., 2022). All these approaches are parameter-efficient, as they only learn a small number of parameters for each language, while the rest remain frozen. However, learning such parameters is not compute-efficient, as it requires a full forward and backward pass over the entire model, including the frozen transformer body.

We introduce *mini-model adaptation*, a new approach to extend MLMs to new languages that is both parameter- and compute-efficient. *Mini-models* are shallow models that are aligned with a

*Work done during an internship at Meta AI
†Work done at Meta AI

Findings of the Association for Computational Linguistics: ACL 2023, pages 5474–5490
July 9-14, 2023

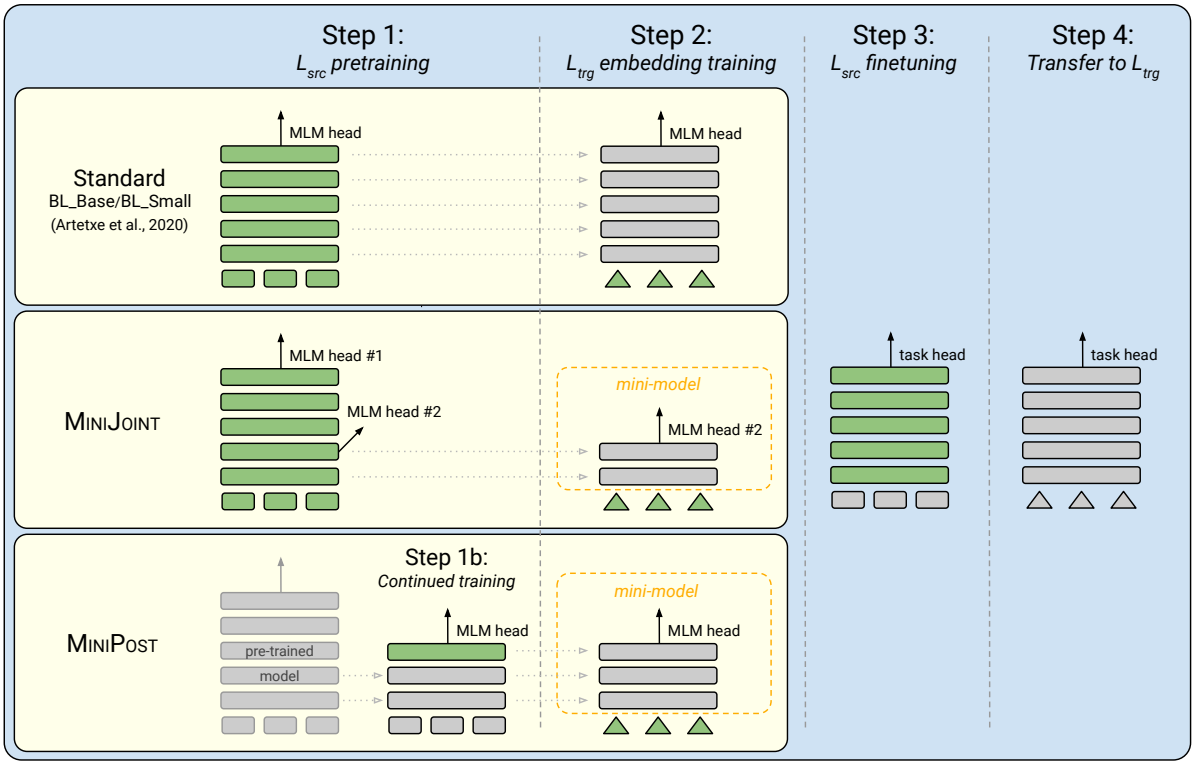


Figure 2: **Standard and mini-model adaptation.** Trainable parameters are *green*, frozen parameters are *gray*. L_{src} embeddings are small rectangles, L_{trg} embeddings are triangles. All approaches use a four-step process for cross-lingual transfer: (1) pretrain an MLM in L_{src}, (2) learn a new embedding layer in L_{trg} via MLM with transformer body frozen, (3) finetune the model in L_{src} with embeddings frozen, (4) zero-shot transfer to L_{trg} by swapping the embeddings. **Standard adaptation (top)** uses the same transformer body for all steps, while our approach learns two aligned models in Step 1—the primary model and a shallower mini-model—and uses the mini-model to learn L_{trg} embeddings efficiently in Step 2. We explore two approaches to learn mini-models: **MiniJoint (center)** jointly pretrains the primary model and mini-model using a secondary MLM head attached at a middle layer; **MiniPost (bottom)** starts from an existing model and builds a mini-model in Step 1b by extracting/freezing a few layers and learning a small number of parameters on top.

larger parent model. Thanks to this, one can efficiently train a new embedding layer for a new language over the mini-model, and plug it directly into the parent for strong cross-lingual performance.

As shown in Figure 2, we explore two approaches to learn mini-models, depending on whether we start from an existing primary model and learn a mini-model posthoc (MiniPost), or we jointly learn the primary model and the mini-model from scratch (MiniJoint). In MiniPost, we extract the bottom layers from the existing MLM, and learn a small number of parameters on top to make it a usable small MLM itself. In the MiniJoint variant, we pretrain an MLM from scratch including a secondary head at a middle layer. Both heads are optimized jointly, creating a complete, well-aligned MLM contained within a larger MLM.

We evaluate our approach on natural language inference (XNLI), question answering (MLQA) and paraphrase identification (PAWS-X). As shown in Figure 1, mini-model adaptation can match the performance of the standard method from Artetxe et al. (2020) using 1.6x and 2.3x less compute for MiniPost and MiniJoint, respectively (averaged over tasks), and retains >98% of performance when trained to completion.

All in all, our work shows that it is possible to adapt language models to new tasks (in this case, new languages) using smaller aligned models for training. While we focus on the problem of cross-lingual lifelong learning to validate this idea, we believe that this new paradigm opens exciting opportunities to make finetuning large language models more affordable.

2 Proposed method

2.1 Standard Adaptation

Artetxe et al. (2020) develop a four-step pipeline for

cross-lingual transfer from a monolingual model, visualized in Figure 2 (top). First, one trains a monolingual MLM in the source language (L_{src}, usually English). Second, the transformer body is frozen, embeddings are re-initialized,[1] and the model is trained with MLM in the target language (L_{trg}). The trainable embeddings are tied with the output projection layer in the MLM head. Third, the L_{src} embeddings are swapped back into the model and frozen, and the transformer body is fine-tuned on the downstream data in L_{src}. Finally, the L_{trg} embeddings are swapped back into the fine-tuned model for zero-shot transfer into L_{trg}. We build two baselines based on this framework: a standard 12-layer (BL_BASE), and a smaller 4-layer version (BL_SMALL).

2.2 Mini-Model Adaptation

Our proposed approach follows a similar four-step training paradigm. However, we learn two aligned models in Step 1: the primary model and a shallow *mini-model*. In Step 2, the L_{trg} embeddings are learned over the mini-model, saving compute with respect to standard adaptation. Steps 3 and 4 are run as usual over the primary model, resulting in a full-size L_{trg} model. For Step 1, we explore the following two alternatives depending on whether we start from an existing L_{src} model, or we are training one from scratch:

MINIJOINT. In this variant, we pretrain a dual-head 12-layer L_{src} transformer from scratch, attaching a secondary head to an intermediary Nth layer (Figure 2, center). The model is trained to minimize the average MLM loss over the two heads. As such, the whole model receives gradient updates from the primary head, and the bottom layers also get updates from the secondary head. Having done that, we extract the bottom N layers and the secondary head to create the mini-model for Step 2. Unless otherwise indicated, we use $N = 4$.

MINIPOST. Here, we start with a regular 12-layer MLM in L_{src} (same as BL_BASE), and build an aligned mini-model in Step 1b (Figure 2, bottom). To that end, we first copy the bottom N layers into a new, shallower model, along with the embeddings and the MLM head. However, this does not work out of the box, as we must bridge the gap between the output of the N bottom layers and the input of the MLM head, which goes through $12-N$ additional layers in the original model. To that end, we add 2 randomly-initialized layers between the N bottom layers and the MLM head, and train them with the MLM objective in L_{src} while keeping the rest of the parameters frozen. Because the new layers are unfrozen, they update to "complete" the MLM—bridging representations from the bottom layers' output to the MLM head's input, and resulting in a mini-model with $N + 2$ layers that is fully functional and aligned with the primary model.

[1]Following Pfeiffer et al. (2021), we initialize the L_{trg} embeddings with overlapping tokens from L_{src} for all methods throughout. Non-overlapping tokens are randomly initialized using the normal distribution with $\mu = 0.0, \sigma = 0.02$.

	GB	Language Family	Word Order	Syn. Dist.	Phylo. Dist.
ar	28.0	Afro-Asiatic: Semitic	SVO/VSO	0.57	1.00
bg	58.0	Indo-European: Slavic	SVO	0.48	0.86
de	67.0	Indo-European: Germanic	SVO/SOV	0.42	0.43
el	47.0	Indo-European: Greek	SVO/VSO	0.52	0.83
en	301.0	Indo-European: Germanic	SVO	0.00	0.00
es	54.0	Indo-European: Romance	SVO	0.40	0.90
fr	57.0	Indo-European: Romance	SVO	0.46	0.90
hi	21.0	Indo-European: Indic	SOV	0.59	0.90
ru	279.0	Indo-European: Slavic	SVO	0.49	0.90
sw	1.7	Niger-Congo: Bantu	SVO	0.57	1.00
th	72.0	Tai-Kadai: Kam-Tai	SVO	0.56	1.00
tr	21.0	Altaic: Turkic	SOV	0.70	1.00
ur	5.7	Indo-European: Indic	SOV	0.67	0.90
vi	138.0	Austro-Asiatic: Viet-Muong	SVO	0.57	1.00
zh	47.0	Sino-Tibetan: Chinese	SVO	0.57	1.00

Table 1: **Languages included in this study.** GB: Size of CC-100 training data in gigabytes. Syn./Phylo. Dist.: syntactic and phylogenetic distance from English, respectively, according to *lang2vec* (Littell et al., 2017).

3 Experimental Settings

Languages and Data. Following common practice, we use English as the source language (L_{src}), and experiment with 14 other languages as the target (L_{trg}). We use CC-100 (Conneau et al., 2020) as our training corpus, which is a filtered version of CommonCrawl. We report the full list of languages along with their corpus size and linguistic details in Table 1. Each language is preprocessed individually using SentencePiece (Kudo and Richardson, 2018) with a vocabulary size of 50,000.

Models. We use the RoBERTa$_{\text{BASE}}$ (Liu et al., 2019) architecture throughout from *fairseq* (Ott et al., 2019). Embeddings are tied. As said in §2, we compare 4 systems: 2 variants of Artetxe et al. (2020) (BL_BASE with 12 layers and BL_SMALL with 4 layers), and 2 variants of our proposed approach where we set $N = 4$ (MINIJOINT, which jointly trains a 12-layer primary model and a 4-layer mini-model from scratch, and MINIPOST,

which starts from a regular 12-layer model and constructs a 6-layer mini-model post-hoc). BL_BASE is a performance upper-bound, as it is the original 12-layer model that is used for adaptation. BL_SMALL is a lower-bound, demonstrating performance of the standard approach using an adaptation model of similar size as ours.

Models are trained for 125,000 steps with a global batch size of 2048, sequence length of 512, and learning rate of 7e-4 with 10,000 warmup updates and linear decay, both for the original pretraining (Step 1), and cross-lingual extension into each language (Step 2). As such, models see 131.1 billion training tokens per language. Step 1b in MINIPOST uses the same training hyperparameters.

Evaluation. We evaluate on 3 tasks: natural language inference in XNLI (Conneau et al., 2018), question answering in MLQA (Lewis et al., 2020), and adversarial paraphrase identification in PAWS-X (Yang et al., 2019). We also report XQuAD (Artetxe et al., 2020) results in §A.2. In all cases, the model is finetuned using the corresponding training data in English (Step 3), and zero-shot transferred into the rest of languages (Step 4). We perform 5 independent finetuning runs with different random seeds, and report average results. During finetuning, we use a peak learning rate of 1e-5 for XNLI and PAWS-X, and 3e-5 for MLQA and XQuAD. Each uses a warmup ratio of 0.06 and linear decay, and is finetuned for 3 epochs.

Estimating FLOPs. We compare training efficiency of different approaches using floating point operations (FLOPs). To calculate FLOPs, we estimate analytically using an adaptation of the formula from Narayanan et al. (2021), detailed in §A.1. When doing so, we exclusively consider the cost of expanding the model to a new language (Step 2), which is the most significant in the cross-lingual lifelong learning setup that our work addresses.[2] We also report NVIDIA V100 GPU training days as a more interpretable number, which we estimate analytically using an estimated throughput of 30 TFLOP/s, or 1 V100 day = 2.592 EFLOPs.

In some of our experiments, we are interested in estimating the training FLOPs required to achieve certain performance. However, this cannot be computed precisely, as we only have a limited number of intermediate checkpoints.[3] For that reason, we identify the checkpoints immediately before and after which the model first scores the desired performance, and use linear interpolation to estimate the step at which the exact score would have been hit. For instance, if MINIPOST obtains an accuracy of 48% at the 5,000 update checkpoint (∼1.17 EFLOPs) and 52% at the 10,000 update checkpoint (∼2.34 EFLOPs), we estimate that the accuracy of 50% was achieved at 7,500 steps (∼1.76 EFLOPs).

4 Main Results

4.1 Performance at Training Completion

Table 2 reports performance at training completion (i.e., after 125,000 updates in Step 2). As expected, BL_BASE obtains the best results, but its training cost is also the highest. In contrast, MINIJOINT requires nearly one third of the compute, while obtaining similar results. More concretely, it is marginally better on PAWS-X, while moderately (1-2 points) worse on MLQA and XNLI. Averaged over tasks, MINIJOINT retains 98.7% of BL_BASE's performance[4] at 39% of its cost. This validates the core hypothesis of our work—learning target language embeddings over the mini-model is almost as effective as learning them over the original model, while being significantly cheaper.

MINIPOST follows a similar trend, retaining 99.3% of BL_BASE's performance at nearly half of its cost. This shows that mini-models do not need to be trained from scratch, but one can take any existing English model and build it's corresponding mini-model post-hoc.

BL_SMALL performs substantially worse than our proposed approach. BL_SMALL has the same training cost as MINIJOINT, but is 4.0 points worse on XNLI, 4.8 points worse on MLQA, and 9.0 points worse on PAWS-X. This shows that our idea of having two aligned models—a shallow one for efficient adaptation and a deep one for best performance at test time—is critical, as using a shallow model both for adaptation and inference performs considerably worse.

[2]While Step 1 can also be expensive, it is amortized over time: the initial model is trained only once, but extended to new languages many times. The cost of Step 1 is similar for BL_BASE and MINIJOINT, as the overhead of the second head is small (∼30.4 vs. ∼32.2 V100 days for a 12-layer model). MINIPOST incurs extra cost from Step 1b, but this is relatively small compared to the cost of pretraining (see §A.1).

[3]We checkpoint every 5000 updates for BL_SMALL, MINIJOINT, MINIPOST. As each step of BL_BASE is more expensive, we checkpoint every 1000 updates for more fine-grained estimates. We save extra checkpoints for MINIJOINT and MINIPOST at steps 1000, 2000, 3000 and 4000 for De, Fr, Es, and Zh, as these adapt rapidly for certain tasks.

[4]$(\frac{70.3}{72.0} + \frac{56.0}{56.9} + \frac{83.5}{83.4})/3 \approx 0.987$

		Train cost		XNLI (acc)														
		EFLOPs	days	ar	bg	de	el	es	fr	hi	ru	sw	th	tr	ur	vi	zh	avg
Standard	BL_BASE	54.1	20.9	70.2	78.4	76.2	76.1	79.2	78.9	65.6	72.5	68.2	70.1	67.1	60.9	72.1	72.4	72.0
	BL_SMALL	21.1	8.1	65.4	71.1	68.0	69.1	71.3	71.0	61.8	66.2	63.8	64.9	64.4	57.9	65.7	67.6	66.3
Proposed	MINIPOST	29.3	11.3	70.1	77.8	75.7	75.5	78.5	78.2	63.9	72.2	67.5	70.2	64.8	59.2	70.8	72.0	71.2
	MINIJOINT	21.1	8.1	69.3	77.6	75.0	74.7	78.4	77.7	62.4	71.7	66.9	68.8	63.7	58.1	69.2	70.8	70.3

(a) XNLI results

		Train cost		MLQA (F1)							PAWS-X (acc)				
		EFLOPs	days	ar	de	es	hi	vi	zh	avg	de	fr	es	zh	avg
Standard	BL_BASE	54.1	20.9	51.2	61.0	66.6	48.5	57.3	56.8	56.9	84.7	85.8	86.0	77.3	83.4
	BL_SMALL	21.1	8.1	46.0	54.6	59.8	43.1	53.2	50.8	51.2	74.2	76.4	76.6	70.8	74.5
Proposed	MINIPOST	29.3	11.3	50.8	60.4	66.7	48.9	56.6	56.7	56.7	83.8	86.2	86.3	76.0	83.1
	MINIJOINT	21.1	8.1	50.8	60.1	66.0	46.4	57.2	55.6	56.0	84.4	85.1	86.7	77.9	83.5

(b) MLQA and PAWS-X results

Table 2: **Performance at training completion.** Both variants of our approach nearly match the performance of BL_BASE at a substantially lower cost, while BL_SMALL significantly lags behind. *days*: V100 GPU days.

4.2 GPU days to Near-Maximal Performance

While we previously compared approaches at training completion, one can also apply early stopping, sacrificing some performance to gain on efficiency. This also allows to compare different approaches head-to-head according to the compute they require to achieve a given score—assuming we stop training as soon as the desired performance is hit. To that end, we fix our target score as 95% of the performance obtained by BL_BASE at the end of training, which we call *near-maximal performance*.[5] Results are in Table 3, and average speedup of our approach over standard adaptation is in Figure 1.[6]

Overall, MINIJOINT does best: when per-language speedup is averaged across languages, we see that it requires about half to one-third the compute of BL_BASE to achieve the same performance in all tasks. MINIPOST has more modest speedups, but is still substantially faster than standard adaptation to hit the desired performance. This shows that, if possible, it is preferable to pretrain mini-models jointly with the primary model, but our approach can also bring substantial speedups when starting with an existing pretrained model.

It is also remarkable that there is a considerable variance across tasks. In particular, all approaches require substantially less compute to achieve the target performance in PAWS-X when compared to XNLI and MLQA. The relative speedup of mini-model adaptation is also considerably higher on PAWS-X. We also observe a high variance across languages, which we analyze in more detail in §5.4.

5 Analysis

5.1 Training Curves

We visualize the training curves of the different approaches in Figure 3. Consistent with our previous findings, we observe that MINIJOINT is usually the leftmost curve—signifying the most rapid adaptation—at the cost of a slightly lower final score. In contrast, BL_BASE is by far the slowest system approaching its peak performance, while BL_SMALL gets stuck at a poor performance compared to other approaches. Finally, we find that all methods adapt rapidly in PAWS-X, which suggests that this tasks might be easier than the others.

5.2 Mini-Model Depth

We recall that the mini-model in MINIJOINT has 4 layers, whereas the one in MINIPOST has 6 (the bottom 4 taken from the primary model + 2 additional ones trained on top). We made this decision early in the development process based on preliminary experiments. In this section, we more systematically study the effect of mini-model depth on ef-

[5] For instance, BL_BASE obtains 70.2 accuracy on Arabic XNLI at training completion, so we set the target performance for Arabic XNLI at $70.2 * 0.95 = 66.7$. Note that this is also the target performance we use for MINIJOINT and MINIPOST, even if their score at training completion is lower.

[6] *Speedup* is the ratio of V100 days to *near-maximal performance* between two methods. For instance, if BL_BASE requires 2.5 V100 days to achieve near-maximal performance in Arabic XNLI and MINIJOINT requires 1.0, the resulting speedup is $2.5/1.0 \approx 2.5$. Note that Figure 1 reports the average ratio across all languages, which is not the same as the ratio of the average V100 days across all languages.

	XNLI															MLQA							PAWS-X				
	ar	bg	de	el	es	fr	hi	ru	sw	th	tr	ur	vi	zh	avg	ar	de	es	hi	vi	zh	avg	de	fr	es	zh	avg
BL_BASE	2.5	1.2	1.1	1.5	0.8	0.8	3.2	1.8	1.4	1.8	**2.3**	8.3	1.0	1.7	2.1	2.6	1.1	0.8	3.4	1.2	1.8	1.8	0.7	0.6	0.6	0.7	0.6
MINIPOST	1.7	0.9	0.8	0.9	0.4	0.4	**2.5**	1.3	1.1	1.3	3.0	6.5	0.8	1.1	**1.6**	1.7	0.8	0.5	**1.7**	0.9	1.2	**1.1**	0.3	0.3	0.3	0.4	0.3
MINIJOINT	**1.0**	**0.5**	**0.5**	**0.6**	**0.3**	**0.3**	5.9	**0.6**	**0.6**	**0.7**	5.3	**5.4**	**0.6**	**0.8**	**1.6**	**1.0**	**0.6**	**0.4**	5.3	**0.5**	**0.9**	1.5	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**

Table 3: **Estimated V100 training days to achieve near-maximal performance.** Near-maximal performance is defined as 95% of the score of BL_BASE at training completion. ↓ is better. BL_SMALL never achieves near-maximal performance, except on XNLI for Turkish (1.7 days) and Urdu (6.4 days).

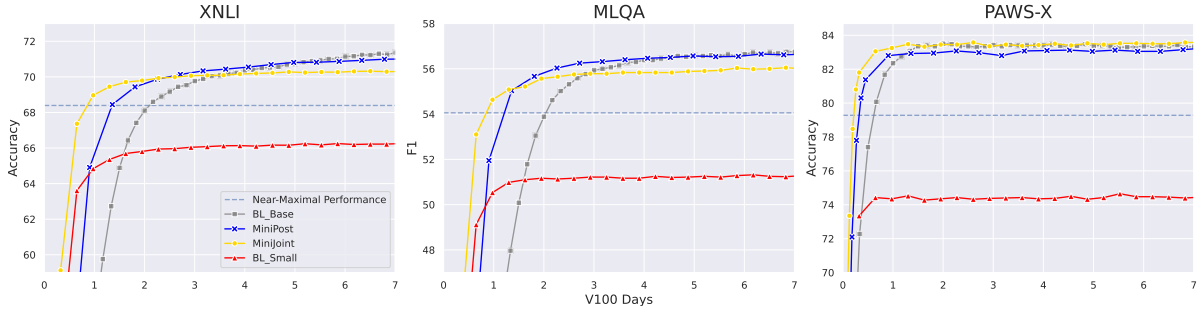


Figure 3: **Training curve through the first GPU-week.** We report XNLI and PAWS-X accuracy and MLQA F1.

ficiency and performance. To that end, we build models with the same architecture as MINIJOINT, but placing the secondary MLM head after layers 2, 6, 10, or 12.[7] We experiment with Arabic, German and Turkish due to compute constraints.

Figure 4 shows the XNLI training curve averaged over 3 languages. We see more rapid adaptation with shallower attachment of the second head, at a cost to final performance. §A.3 shows curves for PAWS-X, MLQA, and XQuAD. For PAWS-X, high performance was rapidly achieved by all models. End-of-training results are in Table A3.

Table 4 reports estimated V100 days to achieve near-maximal performance as defined in §4.2, and upper and lower estimates are in §A.3. We find that the optimal depth of the mini-model is largely language-dependent. Specifically, Arabic and Turkish never hit the target performance with 2 layers, whereas German does so quickly. For Arabic, 4 layers provides the most rapid adaptation, while Turkish requires at least 6. This suggests that it is critical to have some minimum number of layers to achieve good performance, which varies from language to language. But, as long as this minimum is met, shallower mini-models are generally more efficient.

[7] Attaching after layer 12 means that both heads are at the final layer, making it virtually equivalent to BL_BASE.

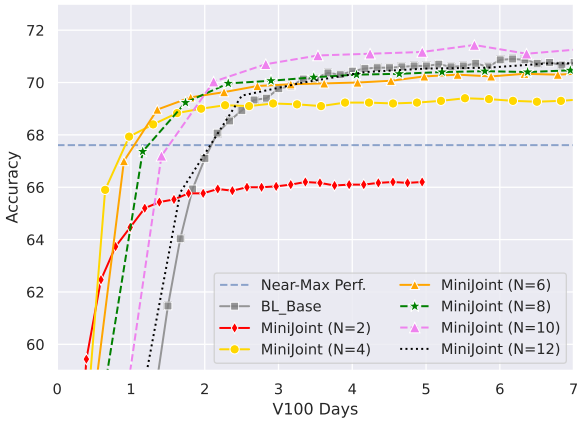


Figure 4: **XNLI training curve for MINIJOINT with secondary head attached at varying layers.** Results are averaged over Arabic, German and Turkish. Final performance is in Table A3.

5.3 English Performance

While all of our results so far correspond to the target languages, we next look into the source language performance. As described in §2.2, MINIPOST uses BL_BASE as the primary model, so their English performance is exactly the same. However, MINIJOINT jointly pretrains the primary model and its aligned mini-model. To understand the effect of the joint pretraining on the monolingual quality of the model, we compare the full MINIJOINT model and its corresponding mini-model with BL_BASE and BL_SMALL. As shown in Table 5, we find that dual-head training does not

	Layer:	2	4	6	8	10	12
de	XNLI	0.4	0.5	0.7	1.0	1.1	1.4
	MLQA	0.7	0.6	0.6	1.1	1.0	1.4
	PAWS	0.2	0.2	0.4	0.5	0.7	1.0
ar	XNLI	∞	1.0	0.9	1.3	1.8	2.4
	MLQA	∞	1.0	1.1	2.2	2.1	2.5
tr	XNLI	∞	5.3	1.5	1.5	1.5	1.6

Table 4: **Estimated V100 days to near-maximal performance (see §4.2) for MINIJOINT with secondary head attached at varying layers.** ↓ better. ∞: never hit target performance.

BL_BASE	86.4
BL_SMALL	79.6
MINIJOINT (full)	86.2
MINIJOINT (mini-model)	79.2

Table 5: **English XNLI accuracy.** §5.3 for details.

damage performance: the full MINIJOINT model performs on-par with the 12-layer baseline, and the 4-layer extracted mini-model performs on-par with the 4-layer baseline.

5.4 Variance Across Languages

While we obtain strong results across the board, there are 3 languages that prove challenging: Hindi, Turkish and Urdu. As shown in Table 3, MINIJOINT takes more than 5 V100 days to achieve near-maximal performance on XNLI for these languages, whereas the rest of the languages require at most 1 day. As seen in §5.2 this can be mitigated by using a deeper mini-model in the case of Turkish. However, we observe that even BL_BASE struggles with Urdu and, to a lesser extent, Hindi. This suggests that there is something making these languages particularly challenging for cross-lingual adaptation, affecting not only our method but also the standard approach from Artetxe et al. (2020).

One hypothesis is that this is due to the high linguistic distance between these languages and English. In Table 1, these are the languages that are the most syntactically distant from English according to *lang2vec*,[8] and the only ones with a pure SOV word order. This is also consistent with German, Spanish and French—the 3 languages that are the closest to English—generally obtaining the fastest adaptation times. In the future, we would like to explore starting with a multilingual model covering a few diverse languages akin to Pfeiffer et al. (2022), which could facilitate adapting to languages that are distant from English but might share features with some of the other languages.

Another potential factor is that Hindi, Turkish and Urdu, along with Swahili, have the smallest training corpora. However, despite having the smallest training corpus with only 1.7GB—∼1/3 the size of Urdu and ∼1/12 of Hindi and Turkish—Swahili exceeds the aforementioned three on both adaptation speed and raw performance on XNLI. Exploring the impact of corpus size was outside of the scope of this work, but we believe that this is an interesting question to address in future work.

6 Related Work

Multilinguality in NLP. One way to create a LM for a particular language is to collect enough data and train from scratch (e.g. Martin et al., 2020; de Vries et al., 2019; Chan et al., 2020). For the majority of languages, however, not enough data exists to train a high-quality model from scratch. Alternatively, one may pretrain a multilingual model on unlabeled data from many languages, which can then be finetuned on labeled data for zero-shot cross-lingual transfer (e.g. Devlin et al., 2019; Conneau and Lample, 2019; Conneau et al., 2020). Multilingual LMs are not without challenges; they are large and expensive to train, suffer from the *curse of multilinguality*, low-resource language performance can lag due to underrepresentation in the training corpus, and they cannot benefit from language-specific tokenization (Conneau and Lample, 2019; Wu and Dredze, 2020; Rust et al., 2021; Doddapaneni et al., 2021, for a survey). Furthermore, not all languages are created alike in multilingual models; Muller et al. (2021) find that some "easy" languages perform well in mBERT out-of-the-box and others are successfully after finetuning with monolingual data, some "hard" languages perform poorly in mBERT even after tuning. Alternatively, one may adapt a pretrained model by finetuning, adding language- or domain-specific adapters (e.g. Rebuffi et al., 2017; Houlsby et al., 2019; Pfeiffer et al., 2022), retraining the lexical embedding layer (Tran, 2020; Artetxe et al., 2020; de Vries and Nissim, 2021), or translating the train, finetuning, or test set (e.g. Wang et al., 2022).

Efficient Adaptation of Language Models. Adapters are a **parameter-efficient** way to extend LMs by training a small number of parameters that can be swapped-in for on-the-fly adaptation at

[8] https://github.com/antonisa/lang2vec

test time as opposed to needing to store full separate models per task or language. Pfeiffer et al. (2020) train small stackable language- and task-specific adapters with respect to a frozen transformer body that is shared between all languages and tasks, allowing simple and quick cross-lingual transfer at test-time. Bapna and Firat (2019) inject adapter layers into a neural machine translation (NMT) model for domain adaptation to obviate the need for full-model finetuning, and use language-specific adapters for high-resource languages to recover from catastrophic forgetting during multilingual NMT training. Alabi et al. (2022) argue that their finetuned mBERT for 17 African languages is parameter efficient because they maintain high-performance with a single model rather than requiring separate models per language. Like Abdaoui et al. (2020), they reduce model size by removing vocabulary tokens not needed for target languages. LoRa adds small trainable matrices corresponding to low-rank decompositions of a weight updates within transformer attention, allowing rapid updates during finetuning (Hu et al., 2022). Prefix-tuning methods are also parameter-efficient (Li and Liang, 2021; Liu et al., 2021).

Compute-efficient methods aim reduce the computation (FLOPs or wall-time) required to train a model. Several authors developed vocabulary adaptation methods which reduce the need to extensively finetune a model or train from scratch (e.g. Chronopoulou et al., 2020; Sachidananda et al., 2021). Though Wang et al. (2020) continued-train mBERT with an extended vocabulary for a new language, convergence is faster than with a bilingual BERT model trained from scratch. Kocmi and Bojar (2020)'s vocabulary adaptation method improves time-to-convergence of a NMT system adapted to a new language. While de Vries and Nissim (2021) learn a new lexical embedding layer on top of GPT-2, which is computationally expensive, they employ engineering strategies to decrease training time, such as 16-bit mixed precision training, reduced window size, and maximum batch size with gradient accumulation. Though they must backpropogate through the entire model during embedding layer relearning, training stabilizes quickly. They adapt larger models by initializing the embedding layer using transformations of embeddings developed on smaller models, noting that the better initialization speeds training.

Variance across languages. Prior work observes similar variation between languages in LM adaptation. When adapting BERT, Tran (2020) see that Hindi showed the slowest growth and lowest final XNLI score of six assessed languages, acknowledging word-order differences. Several authors see performance lags on NLP benchmarks for SOV languages when probing large multilingual models (Doddapaneni et al., 2021, for a review). Pires et al. (2019) find that zero-shot part-of-speech tagging is best when the model has been finetuned on a language that shares word order with the target language. Limisiewicz et al. (2020) attribute the disparity to underrepresentation of SOV languages in the training corpus.

7 Conclusion and Future Work

Our work shows that it is possible to extend pretrained models to new languages using only a fraction of their parameters. We achieve this by learning a new embedding layer over a shallow *mini-model* aligned with the primary model. We explore two approaches to learn mini-models: MINIJOINT augments a transformer with a second MLM head during pretraining, adapting with an average 2.3x speedup over the standard method from Artetxe et al. (2020), and MINIPOST builds a mini-model by extracting a small number of layers from a pretrained model, providing an average 1.6x speedup.

Our analysis reveals that shallower mini-models converge faster but plateau at lower performance. As such, one might explore combining multiple mini-models of different depths, using the shallowest at the beginning of cross-lingual adaptation, and then deeper ones as training progresses. One could add multiple MLM heads to a MINIJOINT model and train all simultaneously to facilitate this.

We would also like to explore applications of mini-model adaptation beyond the multilingual scenario. In particular, by adapting rapidly on models significantly smaller than the base model used for inference, MINIJOINT/MINIPOST might be used to finetune large LMs on modest hardware. This could allow for a new paradigm whereby one shares a small model for adaptation while keeping a large aligned model private behind an API. Clients could then learn parameters for their task on the small model, which are later plugged into the large model for better performance. Shortly after us, Xiao et al. (2023) proposed *Offsite-Tuning*, an adaptation method similar to ours but motivated by privacy.

Limitations

Our study is limited to the adaptation of MLMs to new languages. While we believe that our proposed approach could also be applied more broadly (e.g., autoregressive models instead of MLMs, or adapting to new downstream tasks instead of new languages), further experiments are necessary to empirically verify this. In addition, we observe a considerable variance across languages (§5.4), the reasons for which are not entirely clear. Ideally, we would have a broader set of languages to better study this, as our language set is limited and skewed towards the Indo-European family. Finally, we average results over 5 finetuning runs, but computational restrictions prevented us from also averaging over multiple pretraining runs. As discussed in §A.5, we observed a non-negligible variance over pretraining runs in a preliminary experiment, but a more systematic exploration is necessary to better understand its impact.

Acknowledgements

The authors would like to thank Patrick Littell for helpful discussions about *lang2vec*, along with Philipp Koehn, Elina Baral, Sophia Hager, and Mathias Unberath for helpful discussions and feedback.

References

Amine Abdaoui, Camille Pradel, and Grégoire Sigel. 2020. Load what you need: Smaller versions of mutililingual BERT. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 119–123, Online. Association for Computational Linguistics.

Jesujoba O. Alabi, David Ifeoluwa Adelani, Marius Mosbach, and Dietrich Klakow. 2022. Adapting pre-trained language models to African languages via multilingual adaptive fine-tuning. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 4336–4349, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. 2020. On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4623–4637, Online. Association for Computational Linguistics.

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, Hong Kong, China. Association for Computational Linguistics.

Branden Chan, Stefan Schweter, and Timo Möller. 2020. German's next language model. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6788–6796, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Alexandra Chronopoulou, Dario Stojanovski, and Alexander Fraser. 2020. Reusing a Pretrained Language Model on Languages with Limited Corpora for Unsupervised NMT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2703–2711, Online. Association for Computational Linguistics.

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.

Alexis Conneau and Guillaume Lample. 2019. Cross-lingual language model pretraining. *Advances in neural information processing systems*, 32.

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel Bowman, Holger Schwenk, and Veselin Stoyanov. 2018. XNLI: Evaluating cross-lingual sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2475–2485, Brussels, Belgium. Association for Computational Linguistics.

Wietse de Vries and Malvina Nissim. 2021. As good as new. how to successfully recycle English GPT-2 to make models for other languages. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 836–846, Online. Association for Computational Linguistics.

Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. 2019. Bertje: A dutch bert model. *arXiv preprint arXiv:1912.09582*.

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.

Sumanth Doddapaneni, Gowtham Ramesh, Anoop Kunchukuttan, Pratyush Kumar, and Mitesh M Khapra. 2021. A primer on pretrained multilingual language models. *arXiv preprint arXiv:2107.00676*.

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.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Tom Kocmi and Ondřej Bojar. 2020. Efficiently reusing old models across languages via transfer learning. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pages 19–28, Lisboa, Portugal. European Association for Machine Translation.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Patrick Lewis, Barlas Oguz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. 2020. MLQA: Evaluating cross-lingual extractive question answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7315–7330, Online. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Tomasz Limisiewicz, David Mareček, and Rudolf Rosa. 2020. Universal Dependencies According to BERT: Both More Specific and More General. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2710–2722, Online. Association for Computational Linguistics.

Patrick Littell, David R Mortensen, Ke Lin, Katherine Kairis, Carlisle Turner, and Lori Levin. 2017. Uriel and lang2vec: Representing languages as typological, geographical, and phylogenetic vectors. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, pages 8–14.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. Gpt understands, too. *arXiv preprint arXiv:2103.10385*.

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. *arXiv preprint arXiv:1907.11692*.

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric de la Clergerie, Djamé Seddah, and Benoît Sagot. 2020. CamemBERT: a tasty French language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7203–7219, Online. Association for Computational Linguistics.

Meryem M'hamdi, Xiang Ren, and Jonathan May. 2022. Cross-lingual lifelong learning.

Benjamin Muller, Antonios Anastasopoulos, Benoît Sagot, and Djamé Seddah. 2021. When being unseen from mBERT is just the beginning: Handling new languages with multilingual language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 448–462, Online. Association for Computational Linguistics.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA. Association for Computing Machinery.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Jonas Pfeiffer, Naman Goyal, Xi Lin, Xian Li, James Cross, Sebastian Riedel, and Mikel Artetxe. 2022. Lifting the curse of multilinguality by pre-training modular transformers. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3479–3495, Seattle, United States. Association for Computational Linguistics.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer.

In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2021. UNKs everywhere: Adapting multilingual language models to new scripts. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10186–10203, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy. Association for Computational Linguistics.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. *Advances in neural information processing systems*, 30.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. How good is your tokenizer? on the monolingual performance of multilingual language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, Online. Association for Computational Linguistics.

Vin Sachidananda, Jason Kessler, and Yi-An Lai. 2021. Efficient domain adaptation of language models via adaptive tokenization. In *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing*, pages 155–165, Virtual. Association for Computational Linguistics.

Ke Tran. 2020. From english to foreign languages: Transferring pre-trained language models. *arXiv preprint arXiv:2002.07306*.

Xinyi Wang, Sebastian Ruder, and Graham Neubig. 2022. Expanding pretrained models to thousands more languages via lexicon-based adaptation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 863–877, Dublin, Ireland. Association for Computational Linguistics.

Zihan Wang, Karthikeyan K, Stephen Mayhew, and Dan Roth. 2020. Extending multilingual BERT to low-resource languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2649–2656, Online. Association for Computational Linguistics.

Shijie Wu and Mark Dredze. 2020. Are all languages created equal in multilingual BERT? In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 120–130, Online. Association for Computational Linguistics.

Guangxuan Xiao, Ji Lin, and Song Han. 2023. Offsite-tuning: Transfer learning without full model. *arXiv preprint arXiv:2302.04870*.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

Yinfei Yang, Yuan Zhang, Chris Tar, and Jason Baldridge. 2019. PAWS-X: A cross-lingual adversarial dataset for paraphrase identification. 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 3687–3692, Hong Kong, China. Association for Computational Linguistics.

A Appendix

A.1 Floating Point Operations (FLOPs)

We estimate total FLOPs for training using the formula from Narayanan et al. (2021), amended for RoBERTa without activation recomputation. Like the authors, we omit calculations over biases, activation functions, softmax, and other minor costs. Assume hidden size h, vocabulary size V, number of layers l, token mask probability p, sequence length s, batch size B, and total training updates U, the total FLOPs during training are:

$$72UBslh^2(1 + \frac{s}{6h} + \frac{p}{12l} + \frac{pV}{12hl}) \quad \text{(A1)}$$

Derivation Recall that multiplying $A \in \mathbb{R}^{m\times n}$ by $B \in \mathbb{R}^{n\times p}$ requires $2mnp$ FLOPs. Each transformer layer consists of a multi-head self-attention block and a linear projection. The attention block has four weight matrices $W_q, W_k, W_v, W_o \in \mathbb{R}^{h\times h}$.[9] The input $x \in \mathbb{R}^{s\times h}$ is projected with W_q, W_k and W_v, requiring $2sh^2$ FLOPs each:

$$Q = xW_q \qquad K = xW_k \qquad V = xW_v$$

Self-attention followed by output projection is:

$$(\text{softmax}(\frac{QK^T}{\sqrt{h}})V)W_O$$

Multiplying QK^T and multiplying the result by V both require $2hs^2$ FLOPs. Multiplying with W_O costs $2sh^2$ FLOPs. In sum, there are $8sh^2 + 4hs^2$ FLOPs to compute the forward pass of the attention block. The output of the attention block ($x \in \mathbb{R}^{s\times h}$) is then passed through two linear layers: $F_0 \in \mathbb{R}^{h\times 4h}$ and $F_1 \in \mathbb{R}^{4h\times h}$. These multiplications cost $8sh^2$ FLOPs each, so total FLOPs per layer is:

$$\text{FLOP}_{\text{layer}} = 24sh^2 + 4hs^2$$

The output $x \in \mathbb{R}^{s\times h}$ passes through the MLM head: a dense layer of size $\mathbb{R}^{h\times h}$ for $2sh^2$ FLOPs, and an output projection of size $\mathbb{R}^{h\times V}$ that costs:

$$\text{FLOP}_{\text{outproj}} = 2shV$$

Only masked tokens are passed through MLM head, so the total flops in the LM head is

$$\text{FLOP}_{\text{lm}} = p(2sh^2 + 2shV)$$

[9] We demonstrate the calculation over one head, as using more heads results in the same FLOPs calculation.

In sum, the total estimated FLOPs for a forward pass of RoBERTa with a batch size of 1 is:

$$\begin{aligned} &l(\text{FLOP}_{\text{layer}}) + \text{FLOP}_{\text{lm}} \\ &= l(24sh^2 + 4hs^2) + p(2sh^2 + 2shV) \end{aligned} \quad \text{(A2)}$$

To account for the backward pass, one typically triples the forward pass FLOPs. This is because (1) to backpropogate the error, one calculates the partial derivatives of the loss with respect to the input (activations): $\frac{\partial\delta}{\partial a}$, and (2) to make a weight update, one first must calculate the partial derivatives with respect to the weights: $\frac{\partial\delta}{\partial w}$. Calculating each partial derivative requires the same number of FLOPs as the forward pass, meaning that the backward pass is doubly as expensive.[10] Tripling Equation A2 to account for the backward pass, multiplying by batch size and total updates, and reducing gives Equation A1 for full pretraining.

Adaptation requires an amended equation for the backward pass because layers are frozen (Step 2: L_{trg} embedding training). The trainable embeddings are tied to the output projection layer in the MLM head: thus, trainable input embeddings are passed through frozen layers, which passes through the MLM head consisting of a frozen dense layer and *trainable* output projection. To backpropogate the error to the embeddings, we must (1) calculate $\frac{\partial\delta}{\partial a}$ for the entire model, requiring the same number of FLOPs as the forward pass.[11] Because the MLM head's output projection layer is also trainable, we also calculate $\frac{\partial\delta}{\partial w}$ here on the backward pass. In total, this gives the below equation for Step 2, after multiplying for batch size and total updates:

$$\begin{aligned} &UB(2l\text{FLOP}_{\text{layer}} + 2\text{FLOP}_{\text{lm}} + p\text{FLOP}_{\text{outproj}}) \\ &= 48UBslh^2(1 + \frac{s}{6h} + \frac{p}{12l} + \frac{pV}{8hl}) \end{aligned} \quad \text{(A3)}$$

Thus, adaptation with 4 layers requires ∼21.1 EFLOPs versus ∼29.3 EFLOPs during pretraining. For 12 layers, adaptation requires ∼54.1 EFLOPs versus ∼78.8 in pretraining.

[10] One typically does not add in the cost of the weight update, because this is relatively small.

[11] Some additional backward computation is required here, but we make this simplification.

MINIPOST FLOPs in Step 1b Step 1b of MINIPOST builds small mini-model with embeddings and first l_f layers frozen. These frozen layers do not require the backward pass. Furthermore, the frozen LM head does not require calculating $\frac{\partial \delta}{\partial w}$, only $\frac{\partial \delta}{\partial a}$. Of the trainable layers, each require both $\frac{\partial \delta}{\partial a}$ and $\frac{\partial \delta}{\partial w}$, except the first trainable layer which only needs $\frac{\partial \delta}{\partial w}$ (because it does not pass back the error). Given trainable layers l_t, the total cost for creating the mini-model in MINIPOST is:

$$
\begin{aligned}
&= UB(l(\text{FLOP}_{\text{layer}}) + 2\text{FLOP}_{\text{lm}} + (2l_t - 1)\text{FLOP}_{\text{layer}}) \\
&= UB((l + 2l_t - 1)\text{FLOP}_{\text{layer}} + 2\text{FLOP}_{\text{lm}}) \\
&= UB((l + 2l_t - 1)(24sh^2 + 4hs^2) + 4psh^2 + 4pshV)
\end{aligned} \tag{A4}
$$

Concretely, the cost of training a 6-layer mini-model in this work is ∼21.6 EFLOPs. In comparison, pretraining the vanilla 12-layer RoBERTa base model requires ∼78.8 EFLOPs.

A.2 XQuAD

The Cross-lingual Question Answering Dataset (XQuAD; Artetxe et al., 2020) covers a more extensive set of languages than MLQA. We evaluate the same models tuned for QA in the main body of the paper on XQuAD. Final F1 and V100 days to achieve near-maximal performance are in Tables A1 and A2. We also show the growth curve for F1 through the first V100-week in Figure A1.

	XQuAD										
	ar	de	el	es	hi	ru	th	tr	vi	zh	avg
BL_BASE	2.3	1.1	1.7	0.8	3.3	1.9	2.1	2.2	1.2	1.5	1.8
MINIPOST	1.4	0.8	1.0	**0.4**	**1.3**	1.3	1.4	**1.2**	0.8	1.0	1.1
MINIJOINT	**0.8**	**0.4**	**0.6**	0.5	3.1	**0.6**	**0.6**	∞	**0.5**	**0.6**	**0.9***

Table A1: **Estimated V100 training days to achieve near-maximal performance (see §4.2) on XQuAD.** ∞: never hit target performance. BL_SMALL never achieves near-maximal performance. ∗excludes Turkish, which never hit near-maximal performance.

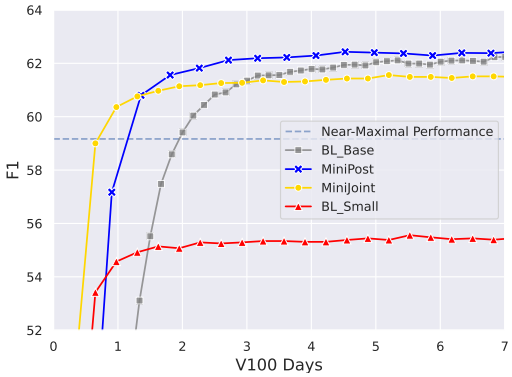


Figure A1: **XQuAD performance in first GPU-week.**

A.3 Mini-Model Depth: MLQA, PAWS-X, and XQuAD

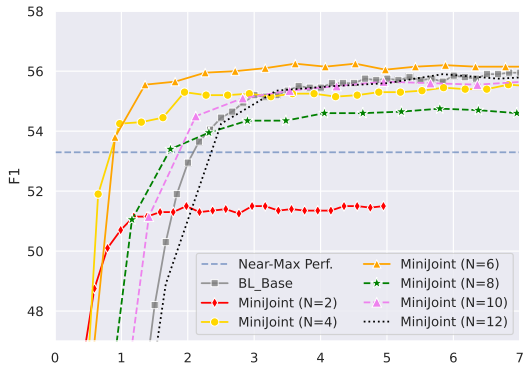
We extend the results of §5.2 to MLQA, PAWS-X, and XQuAD, shown in Figure A2. Figure A3 shows training curves for the particularly challenging language of Turkish on XNLI and XQuAD. Table A3 shows performance at training completion.

	Figure:	**Fig. 4**	**A2(a)**	**A2(b)**	**A2(c)**	**A3(a)**	**A3(b)**
***N* =**	**2**	66.2	51.5	83.8	54.1	58.3	46.8
	4	69.3	55.4	84.4	59.0	63.7	53.2
	6	70.3	56.2	83.7	59.8	65.6	56.0
	8	70.6	54.8	83.3	58.6	66.7	55.0
	10	71.4	55.8	82.9	58.8	67.9	53.2
	12	71.1	56.0	81.5	60.3	68.5	57.4
	BL_Base	71.2	56.1	84.7	59.6	67.1	57.2

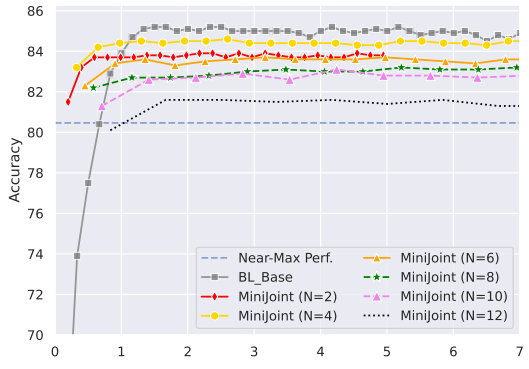
Table A3: **Performance at end of training for MINIJOINT.** Results correspond to Figures 4, A2, and A3.

		Train cost		XQuAD (acc)										
		EFLOPs	days	ar	de	el	es	hi	ru	th	tr	vi	zh	avg
Standard	BL_BASE	54.1	20.9	52.4	69.2	70.0	74.8	53.8	68.7	57.1	57.2	65.9	53.7	62.3
	BL_SMALL	21.1	8.1	48.2	61.2	63.3	65.8	46.2	61.0	50.4	52.1	60.3	46.2	55.5
Proposed	MINIPOST	29.3	11.3	52.4	68.8	69.9	75.1	55.3	68.2	56.9	58.6	65.7	53.5	62.4
	MINIJOINT	21.1	8.1	53.8	70.1	69.9	73.7	51.8	68.6	56.5	53.2	64.9	52.5	61.5

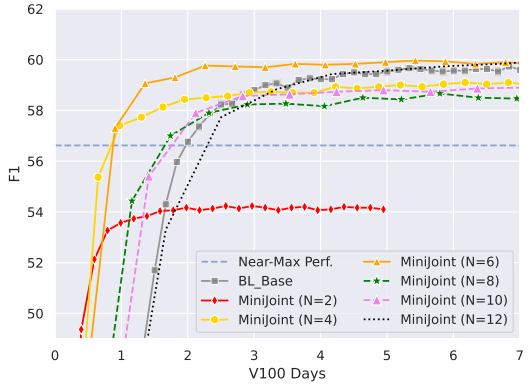
Table A2: **XQuAD performance at training completion.** Both variants of our approach nearly match the performance of BL_BASE at a substantially lower cost, while BL_SMALL significantly lags behind.



(a) **MLQA.** (Arabic, German)

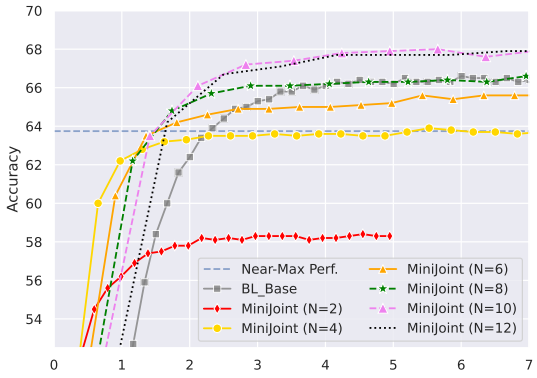


(b) **PAWS-X.** (German only)

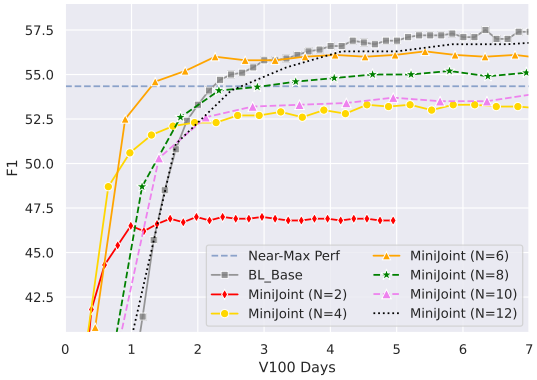


(c) **XQuAD.** (Arabic, German, Turkish)

Figure A2: **Training curves for MINIJOINT with secondary head attached at varying layers.** Averaged over languages. Final performance in Table A3.



(a) **XNLI.**



(b) **XQuAD.**

Figure A3: **Training curves for MINIJOINT with secondary head attached at varying layers.** Turkish. Final performance in Table A3.

	Layers:	2	4	6	8	10	12
de	XNLI	0.4 (0.2 - 0.4)	0.5 (0.3 - 0.7)	0.7 (0.5 - 0.9)	1.0 (0.6 - 1.2)	1.1 (0.7 - 1.4)	1.4 (0.8 - 1.7)
	XQUAD	0.4 (0.2 - 0.4)	0.4 (0.3 - 0.7)	0.6 (0.5 - 0.9)	0.9 (0.6 - 1.2)	0.7 (0.7 - 1.4)	1.3 (0.8 - 1.7)
	MLQA	0.7 (0.6 - 0.8)	0.6 (0.3 - 0.7)	0.6 (0.5 - 0.9)	1.1 (0.6 - 1.2)	1.0 (0.7 - 1.4)	1.4 (0.8 - 1.7)
	PAWS	0.2 (0.0 - 0.2)	0.2 (0.2 - 0.3)	0.4 (0.0 - 0.5)	0.5 (0.0 - 0.6)	0.7 (0.0 - 0.7)	1.0 (0.8 - 1.7)
ar	XNLI	∞	1.0 (0.7 - 1.0)	0.9 (0.9 - 1.4)	1.3 (1.2 - 1.7)	1.8 (1.4 - 2.1)	2.4 (1.7 - 2.5)
	XQUAD	∞	0.8 (0.7 - 1.0)	0.9 (0.5 - 0.9)	1.6 (1.2 - 1.7)	1.8 (1.4 - 2.1)	2.4 (1.7 - 2.5)
	MLQA	∞	1.0 (1.0 - 1.3)	1.1 (0.9 - 1.4)	2.2 (1.7 - 2.3)	2.1 (1.4 - 2.1)	2.5 (1.7 - 2.5)
tr	XNLI	∞	5.3 (5.2 - 5.5)	1.5 (1.4 - 1.8)	1.5 (1.2 - 1.7)	1.5 (1.4 - 2.1)	1.6 (0.8 - 1.7)
	XQUAD	∞	∞	1.3 (0.9 - 1.4)	3.0 (2.9 - 3.5)	∞	2.7 (2.5 - 3.3)

Table A4: **Estimated V100 days to near-maximal performance (see §4.2) for MINIJOINT with secondary head attached at varying layers.** ↓ better. ∞: never hit target performance. (lower - upper) bounds on the estimate.

A.4 Upper/Lower estimates on Time to Near-Maximal Performance

In §4.2, we use linear interpolation to estimate GPU days to near-maximal performance if target performance occurred between checkpoints. In Table A4, we show the upper and lower estimates. Models are checkpointed every 5000 updates, so a lower estimate of 0.0 implies that the target score was achieved before first checkpoint. Because MINIJOINT with the secondary head attached at layer 4 was part of the main experiments, it was also checkpointed on steps 1000, 2000, 3000, and 4000. As such, estimates lower than 0.3 from this model imply that the target score was achieved hit before step 5000 (the first checkpoint for other models).

A.5 Variance across Pretraining Runs

While we average results over 5 finetuning runs, we always use the same pretrained model. Early in development, we noticed that there could be a difference between different pretraining runs. While it was not feasible to repeat all experiments with different pretraining seeds due to computational cost, we performed 3 additional runs of BL_BASE for Arabic. We see a difference up to 3 points across runs in Table A5. This is task dependent, as the best run on XNLI is the worst on MLQA.

	XNLI	MLQA
Run #1	67.7	51.4
Run #2	69.3	51.1
Run #3	68.7	**52.7**
Main run	**69.6**	49.6

Table A5: **Arabic development performance for BL_BASE with different pretraining seeds.** Results averaged over 5 finetuning runs.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Limitations

☑ A2. Did you discuss any potential risks of your work?
Limitations

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract/Intro

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
Not applicable. Left blank.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Not applicable. Left blank.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Not applicable. Left blank.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Not applicable. Left blank.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Not applicable. Left blank.

C ☑ Did you run computational experiments?

Sections 4, 5, Appendix

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
V100 GPU days are reported throughout. We cite the base architecture and report architectural changes.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 3

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Sections 4 & 5

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Section 3

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Not applicable. Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Not applicable. Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Not applicable. Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Not applicable. Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Not applicable. Left blank.